



---

## Towards Robust Multiagent Plans

Michal Pechoucek  
CZECH TECHNICAL UNIVERSITY IN PRAGUE

---

01/20/2016  
Final Report

DISTRIBUTION A: Distribution approved for public release.

Air Force Research Laboratory  
AF Office Of Scientific Research (AFOSR)/ IOE  
Arlington, Virginia 22203  
Air Force Materiel Command

<b>REPORT DOCUMENTATION PAGE</b>				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Executive Services, Directorate (0704-0188). Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.</p>					
<b>1. REPORT DATE (DD-MM-YYYY)</b> 20-01-2016		<b>2. REPORT TYPE</b> Final		<b>3. DATES COVERED (From - To)</b> 01-06-2012 to 31-05-2015	
<b>4. TITLE AND SUBTITLE</b> Towards Robust Multiagent Plans				<b>5a. CONTRACT NUMBER</b>	
				<b>5b. GRANT NUMBER</b> FA8655-12-1-2096	
				<b>5c. PROGRAM ELEMENT NUMBER</b> 61102F	
<b>6. AUTHOR(S)</b> Michal Pechoucek				<b>5d. PROJECT NUMBER</b>	
				<b>5e. TASK NUMBER</b>	
				<b>5f. WORK UNIT NUMBER</b>	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> CZECH TECHNICAL UNIVERSITY IN PRAGUE ŽIKOVA 4 PRAGUE 6, 166 36 CZ				<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> EOARD Unit 4515 APO AE 09421-4515				<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b> AFRL/AFOSR IOE	
				<b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b>	
<b>12. DISTRIBUTION/AVAILABILITY STATEMENT</b> A DISTRIBUTION UNLIMITED: PB Public Release					
<b>13. SUPPLEMENTARY NOTES</b>					
<b>14. ABSTRACT</b> <p>This project, titled Towards Robust Multiagent Plans with its extension Domain-Independent Multiagent Planning: Models, Stability, and Complexity focused on theoretical and applied research in field of multiagent planning in dynamic environments. The objective of the project was to connect, both formally and empirically, the developments in domain-specific multiagent planning and the concepts of generic, domain-independent problem solving and propose solutions and techniques robust to uncertainty, dynamism and non-determinism of environments modeled closely to real world. This final report summarizes general overview of the project, achievements of the project mostly in the form of published research work, and describes the demonstrator. The research work in the project comprised of four accepted and one submitted journal publication aiming at advanced techniques for multiagent plan repair, simple regret optimization in online planning, oversubscription planning, and a submitted journal article on novel type of multi-heuristic search for multiagent planning together with overview of the Multiagent Distributed and Local Asynchronous (MADLA) planner. Furthermore, eight accepted papers at the top artificial intelligence and planning conferences focused on fault tolerant planning and interruptible exploration technique usable in Monte-Carlo tree search algorithms. The submitted and accepted workshop papers provided good ground for valuable discussion at the specific research forums and propagated the work.</p>					
<b>15. SUBJECT TERMS</b> EOARD, Multi-agent systems, Planning					
<b>16. SECURITY CLASSIFICATION OF:</b>		<b>17. LIMITATION OF ABSTRACT</b>	<b>18. NUMBER OF</b>		

<b>a. REPORT</b>	<b>b. ABSTRACT</b>	<b>c. THIS PAGE</b>	SAR	<b>PAGES</b>	<b>19a. NAME OF RESPONSIBLE PERSON</b> Michal Pechoucek
Unclassified	Unclassified	Unclassified			<b>19b. TELEPHONE NUMBER</b> <i>(Include area code)</i> 420-22435 7355

**FA8655-12-1-2096**

# **Towards Robust Multiagent Plans**

## **Final Report**

December 31, 2015

**Agent Technology Center,  
Department of Computer Science and Engineering,  
Faculty of Electrical Engineering  
Czech Technical University in Prague**



**Faculty of Industrial Engineering & Management,  
Technion - Israel Institute of Technology**



Antonín Komenda, Michal Štolba  
Principal Investigators: Carmel Domshlak, Michal Pěchouček



## Executive summary

The project (FA8655-12-1-2096) *Towards Robust Multiagent Plans* with its extension *Domain-Independent Multiagent Planning: Models, Stability, and Complexity* focused on theoretical and applied research in field of multiagent planning in dynamic environments. The objective of the project was to connect, both formally and empirically, the developments in domain-specific multiagent planning and the concepts of generic, domain-independent problem solving and propose solutions and techniques robust to uncertainty, dynamism and non-determinism of environments modeled closely to real world. The project was planned for three years. In Y3, it *was prolonged to 43 months*, that is till the end of 2015.

This final report summarizes general overview of the project, achievements of the project mostly in the form of published research work, and describes the demonstrator.

The research work in the project comprised of four accepted and one submitted journal publication aiming at advanced techniques for multiagent plan repair, simple regret optimization in online planning, oversubscription planning, and a submitted journal article on novel type of multi-heuristic search for multiagent planning together with overview of the Multiagent Distributed and Local Asynchronous (MADLA) planner. Furthermore, eight accepted papers at the top artificial intelligence and planning conferences focused on fault tolerant planning and interruptible exploration technique usable in Monte-Carlo tree search algorithms. The submitted and accepted workshop papers provided good ground for valuable discussion at the specific research forums and propagated the work.

In the third year of the project, we have organized the first international Competition of Distributed and Multiagent Planners (CoDMAP) which helped to unify implementation of the multiagent planners in the multiagent planning research community and to propagate multiagent planning in general.

The project demonstrator developed in the first and last years of the project is based on a simulation system developed in the Tactical AgentFly and Tactical AgentScout projects (funded previously by US ARMY, CERDEC). The tactical environment used is a high-fidelity multiagent model which utilizes the distributed multiagent planner and integrates the designed multiagent online planning algorithm for static and dynamic tactical scenarios.

The report is closed up with the publications providing details on the particular problems solved during the project.



# 1 Overview

Achieving joint objectives by teams of cooperative planning agents requires significant reasoning but also coordination and communication efforts especially in case of planning in dynamic environments. The robustness of the resulting plans with respect to unforeseen changes is a key property. The research fields tackling such problems are uncertainty planning, continuous and online planning, plan repairing and probabilistic planning. This project focuses on study of various approaches from these fields and the key objective is to extend the approaches to multiagent setting and therefore to area of distributed problem solving in general. The research targets comprise of:

**RT1: Formal modeling.** Formalization of multiagent planning problems towards robustness against the uncertainty and dynamism in the environment. Its mathematical models, specification of descriptive languages for representing the problems and formal study of computational complexity, interaction stability and other properties of robust multi-agent planning and multi-agent plans.

**RT2: State of the art study.** Study prior art for existing multiagent planning approaches especially in fields of uncertainty planning, continuous and online planning, plan repairing and probabilistic planning and select/extend the approach best fitting the problem of robust multiagent planning.

**RT3: Algorithm design.** Research review of existing planning and coordination algorithms matching the multiagent planning problem definition. Extension of existing planning methods (heuristic-based, sampling-based, plan adaptation-based) for robust multiagent planning. Suggest and develop various multiagent planning methods optimizing the selected planning criteria of the robust planning.

**RT4: Theoretical analysis.** Perform theoretical analysis of complexity metrics as computational complexity and communication complexity of the developed methods, study relationship between robust planning and multiagent planning techniques.

**RT5: Empirical Analysis.** Experimental analysis of the properties of the designed planning methods, validation of the theoretical results, analysis of their practical usability and generality. For this purpose we will develop a set of benchmark problems that will be used for analysis of the performance of the algorithms and deploy the algorithms into a high-fidelity simulation. The simulation system will be based on the Tactical AgentFly and Tactical AgentScout projects (funded previously by US ARMY, CERDEC) as an experimental testbed. Tactical Environment is a high-fidelity, large scale multiagent model of surveillance and tracking missions executed by a fleet of unmanned aerial vehicles.



## 2 Achievements

The research achievements (RT1, RT2, RT3 and RT4) of the project were reported in form of 4 accepted and 1 submitted journal articles, 8 accepted conference papers and several workshop papers. For the empirical analysis of RT5, we organized a competition of distributed and multiagent planners and developed a demonstrator utilizing the techniques described in the respective papers. This section summarizes the publications, gives an overview of the competition, and describes the demonstrator.

### 2.1 Journal Publications

**Simple Regret Optimization in Online Planning for Markov Decision Processes** In online planning in Markov decision processes (MDPs), the agent focuses on its current state only, deliberates about the set of possible policies from that state onwards and, when interrupted, uses the outcome of that exploratory deliberation to choose what action to perform next. The performance of algorithms for online planning is assessed in terms of *simple regret*, which is the agent’s expected performance loss when the chosen action, rather than an optimal one, is followed. The state-of-the-art algorithms for online planning in general MDPs were either best effort, or guaranteed only polynomial-rate reduction of simple regret over time. We have introduced a new Monte-Carlo tree search algorithm, BRUE, that guarantees *exponential-rate* reduction of simple regret and error probability. This algorithm is based on a simple yet non-standard state-space sampling scheme, MCTS2e, in which different parts of each sample are dedicated to different exploratory objectives. Our empirical evaluation showed that BRUE not only provides superior performance guarantees, but is also very effective in practice and favorably compares to state-of-the-art online planning techniques.

**Deterministic Oversubscription Planning as Heuristic Search: Abstractions and Reformulations** The objective of oversubscription planning (OSP) is to achieve as valuable as possible subset of goals within a fixed allowance of the total action cost. Tracing the key sources of progress in classical planning, we identified a severe lack of effective domain-independent approximations for OSP. With our focus on optimal planning, two classes of approximation techniques have been found especially useful in the context of optimal classical planning: those based on state-space abstractions and these based on logical landmarks for goal reachability. The question we studied was whether some similar-in-spirit, yet possibly mathematically different, approximation techniques can be developed for OSP. In the context of abstractions, we defined the notion of additive abstractions for OSP, study the complexity of deriving effective abstractions from a rich space of hypotheses, and revealed some substantial, empirically relevant islands of tractability. Our empirical evaluation confirmed the effectiveness of the proposed techniques.

**Domain-independent Multiagent Plan Repair** Achieving joint objectives in distributed domain-independent planning problems by teams of cooperative agents requires significant coordination and communication efforts. For systems facing a plan failure in a dynamic environment, arguably, attempts to repair the failed plan in general, and especially in the worst-case scenarios, do not straightforwardly bring any benefit in terms of time complexity. However, in multiagent settings, the communication complexity is of a much higher importance, possibly a high communication overhead might even be prohibitive in certain domains.

We have formally introduced a notion of *multiagent plan repair problem*. Building upon the formal treatment, we have designed three algorithms for multiagent plan repair reducing the problem to specialized instances of the multiagent planning problem. Finally, we have presented an experimental validation, results of which showed that in decentralized systems, where frequent coordination is required to achieve joint objectives, attempts to repair failed multiagent plans leads to lower communication overhead than replanning from scratch.

**Multiagent Plan Repair by Combined Prefix and Suffix Reuse** The plan repairing techniques from the previous article were generalized and the generalization extensively experimentally and theoretically analyzed.

Provided that agents act in an uncertain and dynamic environment, their plans can fail. The straightforward approach to recover from such situations is to compute a new plan from scratch, that is to replan. Even though, in a worst case, plan repair or plan re-use does not yield an advantage over replanning from scratch, there is a sound evidence from practical use that approaches trying to repair the failed original plan can outperform replanning in selected problems. One of the possible plan repairing techniques is based on preservation of fragments of the older plans. The article theoretically analyzed complexity of plan repairing approaches based on preservation of fragments of the original plan and experimentally studied three practical aspects affecting its efficiency in various multiagent settings in a generalized multiagent plan repairing scheme.

We focused both on the computational, as well as the communication efficiency of plan repair in comparison to replanning from scratch and we reported on the influence of the following properties on the efficiency of plan repair: (1) the number of involved agents in the plan repairing process, (2) inter-dependencies among the repaired actions, and finally (3) particular modes of re-use of the older plans.

**The MADLA Planner: Multiagent Planning by Combination of Distributed and Local Heuristic Search** We have prepared an Artificial Intelligence Journal submission summarizing and extending the work on the Multiagent Distributed and Local Asynchronous (MADLA) planner.

Multiagent planning for the multiagent STRIPS model requires a process of distributed plan generation for a cooperative team of agents. Heuristic multiagent state-space search is an obvious candidate providing scheme both for agents' local search and inter-agent proto-

col for distribution of the search. However, distributed heuristic estimation, application of multi-heuristic search and efficient utilization of the decentralized computation power was still an open challenge. The MADLA planner runs a distributed variant of a state-space forward-chaining multi-heuristic search with two versions of a well known FastForward relaxation heuristic, one estimating the particular agent’s local subproblem and another estimating the global heuristic values. We proposed a general asynchronous scheme for such multi-heuristic multiagent searches and provided proofs of soundness and completeness. The asynchronism allowed efficient utilization of agents’ computation power while waiting for responses from other agents. Also, we provided a novel distribution scheme for the FastForward heuristic, inspired by the Set-Additive variation of the FastForward heuristic and with lazy computation of partial relaxed plans. We experimentally compared the proposed multi-heuristic scheme and the two used heuristics per se. The results showed the proposed solution outperforms search with either one of the heuristics used separately, positively combining benefits of both heuristics. In the detailed experimental analysis, we showed limits of the planner and of the used heuristics based on particular properties of the benchmark domains. In a comprehensive set of multiagent planning domains and problems, we demonstrated that the MADLA Planner outperforms all state-of-the-art MA-STRIPS multiagent planners.

## 2.2 Conference Publications

**Monte-Carlo Tree Search: To MC or to DP?** In the context of BRUE research, state-of-the-art Monte-Carlo tree search algorithms can be parametrized with any of the two information updating procedures: Monte-Carlo-backup (MC-) and Dynamic-Programming-backup (DP-backup). The dynamics of these two procedures is very different, and so far, their relative pros and cons have been poorly understood. Formally analyzing the dependency of MC- and DP-backups on various parameters of Markov Decision Processes, we revealed numerous important issues that got hidden by the worst-case bounds on the algorithm performance, and reconfirmed these findings by a systematic experimental test.

**On MABs and Separation of Concerns in Monte-Carlo Planning for MDPs** In the area of online planning and sampling methods, we extended the proposed algorithm BRUE towards problems described as Markov Decision Processes with their special case of stochastic multi-armed bandit problems. We analyzed three state-of-the-art Monte-Carlo tree search algorithms: UCT, BRUE, and MaxUCT. Using the outcome, we (i) introduced two new MCTS algorithms, MaxBRUE, which combines uniform sampling with Bellman backups, and MpaUCT, which combines UCB1 with a novel backup procedure, (ii) analyzed them formally and empirically, and (iii) showed how MCTS algorithms can be further stratified by an exploration control mechanism that improves their empirical performance without harming the formal guarantees.

**Abstractions for Oversubscription Planning** In deterministic oversubscription planning (OSP), the objective is to achieve an as valuable as possible subset of goals within a fixed allowance of the total action cost. Although numerous applications in various fields share this objective, no substantial algorithmic advances have been made beyond the very special settings of net-benefit optimization. Tracing the key sources of progress in classical planning, we have identified a severe lack of domain-independent approximations for OSP, and started with investigating the prospects of abstraction approximations for this problem. In particular, we have defined the notion of additive abstractions for OSP, studied the complexity of deriving effective abstractions from a rich space of hypotheses, and revealed substantial, empirically relevant islands of tractability.

**Landmarks in Oversubscription Planning** Another extension of the OSP direction was pursuing of exploitation of concept of landmarks from classical planning. We developed a framework for exploiting such landmarks in heuristic-search OSP. We showed how standard landmarks of certain classical planning tasks can be compiled into the OSP task of interest, resulting in an equivalent OSP task with a lower budget, and thus with a smaller search space. We then showed how such landmark-based task enrichment can be combined in a mutually stratifying way with the Best-First-Branch-and-Bound search used for OSP planning. Our empirical evaluation confirmed the effectiveness of the proposed landmark-based budget reduction scheme.

**Relaxation Heuristics for Multiagent Planning** In multiagent planning, heuristics significantly improve efficiency of search-based planners, similarly to classical planners. Heuristics based on solving a relaxation of the original planning problem were already intensively studied in context of classical planning. In particular, a popular heuristics is the delete relaxation, where all delete effects of actions are omitted. We have designed and presented a unified view on distribution of delete relaxation heuristics for multiagent planning. We experimentally evaluated properties of the distribution of additive, max and Fast-Forward relaxation heuristics in the implementation of the early versions of the MADLA planner. The best performing distributed relaxation heuristics favorably compared to a state-of-the-art multiagent STRIPS planner in terms of benchmark problem coverage. Finally, we analyzed impact of limited agent interactions by means of recursion depth of the heuristic estimates.

**Fault Tolerant Planning: Complexity and Compilation** In the context of modeling and reasoning about agent actions, contingent and classical planning can often be respectively seen as adopting “extreme pessimism” and “extreme optimism” about the action outcomes. For many everyday scenarios of human reasoning (and thus for many types of autonomous systems), both these approaches are just too extreme. We have examined a planning model that interpolates between classical and contingent planning via tolerance to arbitrary  $\kappa$  faults occurring during plan execution. We have shown that an important fragment of

this *fault tolerant planning (FT-planning)* exhibits both an appealing solution structure, as well as appealing worst-case time-complexity properties. We have also showed that such FT-planning tasks can be efficiently compiled into classical planning as long as the number of possible faults per operator is bounded by a constant, and we have shown that this compilation can be attractive in practice.

**On Combinatorial Actions and CMABs with Linear Side Information** Since classical on-line planning algorithms are typically a tool of choice for dealing with sequential decision problems in combinatorial search spaces, as in multiagent setting, many such problems, however, also exhibit combinatorial actions, yet standard planning algorithms do not cope well with this type of “the curse of dimensionality.” Based on previous work on combinatorial multi-armed bandit (CMAB) problems, we proposed a novel CMAB planning scheme, as well as two specific instances of this scheme, dedicated to exploiting what is called linear side information. Using a representative strategy game as a benchmark, we showed that the resulting algorithms very favorably compete with the state-of-the-art.

**On Interruptible Pure Exploration in Multi-Armed Bandits** Interruptible pure exploration in multi-armed bandits (MABs) is a key component of Monte-Carlo tree search algorithms for sequential decision problems. We introduced Discriminative Bucketing (DB), a novel family of strategies for pure exploration in MABs, which allows for adapting recent advances in non-interruptible strategies to the interruptible setting, while guaranteeing exponential-rate performance improvement over time. Our experimental evaluation demonstrated that the corresponding instances of DB favorably compete both with the currently popular strategies UCB1 and  $\epsilon$ -Greedy, as well as with the conservative uniform sampling.

## 2.3 Workshop Publications

**Fast-Forward Heuristic for Multiagent Planning** Use of heuristics in search-based domain-independent deterministic multiagent planning is as important as in classical planning. We have proposed a formal and an algorithmic adaptation of a well-known heuristic Fast-Forward into multiagent planning. Such treatment is important as it solves challenges in decentralization of this and other heuristics based on relaxation of the original planning problem. Such decentralization enables global heuristic estimates to be computed without exposing local information. Additionally, since Fast-Forward heuristic is based on relaxed planning, we have proposed a multiagent approach for building factored relaxed planning graphs among the agents.

**How to Repair Multiagent Plans: Experimental Approach** Deterministic domain-independent multiagent planning is an approach to coordination of cooperative agents with joint goals. Provided that the agents act in an imperfect environment, such plans can fail. One of the possible techniques how to recover from such failures is to repair the

original plan. One family of such repairing techniques is based on preservation of plan parts. We have experimentally studied three aspects affecting efficiency of plan repairing approaches based on preservation of fragments of the original plan in a multiagent setting with focus both on the computational, as well as the communication efficiency of plan repair in comparison to replanning from scratch.

**On Robustness of CMAB Algorithms: Experimental Approach** Similarly to the most prominent approaches for online planning with polynomial number of possible actions, state-of-the-art algorithms for online planning with exponential number of actions are based on Monte-Carlo sampling. However, without a proper selection of the appropriate subset of actions these techniques cannot be used. The most recent algorithms tackling this problem utilize an assumption of linearity with respect to the combinations of the actions. In this paper, we experimentally analyzed robustness of two state-of-the-art algorithms NMC and LSI for online planning with combinatorial actions in various setups of Real-Time and Turn-Taking Strategy games.

## 2.4 Competition of Distributed and Multiagent Planners (CoDMAP)

As a part of the workshop on Distributed and Multiagent Planning (DMAP) at the International Conference on Automated Planning and Scheduling (ICAPS) 2015, we have organized a competition in distributed and multiagent planning. The main aims of the competition were to consolidate the planners in terms of input format; to promote development of multiagent planners both inside and outside of the multiagent research community; and to provide a proof-of-concept of a potential future multiagent planning track of the International Planning Competition (IPC). In this section we summarize course and highlights of the competition.

**Introduction and Aims of CoDMAP** Various forms of multiagent planning have recently found their way to the automated planning research community, nevertheless, there was no competition of multiagent planners in the tradition of IPC yet. As the organizers of DMAP'15, we have decided to run a co-located competition of multiagent planners.

We chose an approach similar to that of classical planning competitions, to start with the smallest possible subset of features and possibly extend them in the future. One of the main focuses of the competition design was to allow as many existing planners as possible to enter without large-scale modifications. In order to foster our awareness of the existing planners and their possible extensions, we have conducted a public poll<sup>1</sup>. Out of the poll and other considerations arose three main restrictions of the multiagent planning model: deterministic, non-durative actions, full observability (with respect to privacy), cooperative agents and offline planning.

---

<sup>1</sup>The poll form can be found at: <http://bit.ly/1IsNoqY>

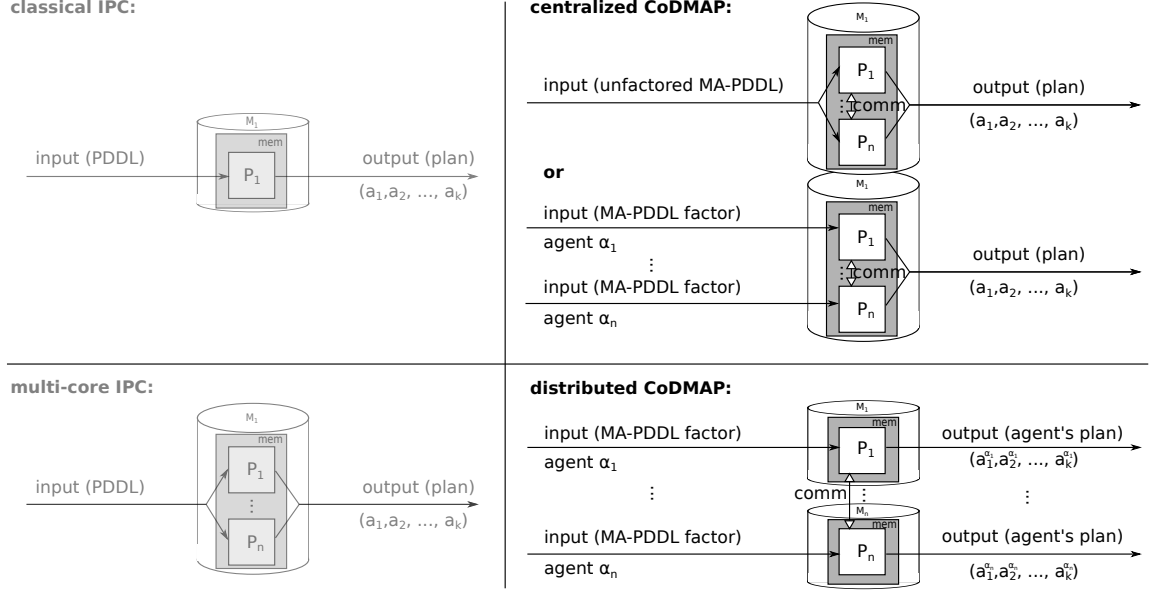


Figure 1: Comparison of IPC and CoDMAP tracks.

**Formalism and Input Language** A crucial point of the competition was to determine a formalism and an input language. For the formalism, we have chosen MA-STRIPS [1] for its simplicity and wide acceptance among existing planners. MA-STRIPS extends the STRIPS formalism with two concepts: (i) *factorization* and (ii) *privacy*. Factorization is defined in the planning problem and prescribes what STRIPS actions can be executed by which agents. A STRIPS fact is private if it is not affected and cannot affect more than one agent.

Following the minimalistic extension of STRIPS to MA-STRIPS, we wanted a simple extension of the common planning language PDDL [7] towards multiagent planning, also compatible with MA-STRIPS. After analysis of candidate languages, we have decided to extend MA-PDDL [6]. The extension<sup>2</sup> came in two flavors, a *factored* description, which allowed the definition of separate domain and problem description for each agent, and an *unfactored* description, which allowed the definition of factorized privacy in a single domain and problem description. Additionally, our generalized definition of privacy was enough to comprise MA-STRIPS privacy, but allowed for more general definitions, possibly usable in future multiagent planning competitions.

**Competition Tracks** A success of a planning competition is determined to large extent by the number of contestants and as there was no historical experience from previous multiagent planning competitions, we wanted to open the competition to the widest possible

<sup>2</sup>The extended BNF can be found at <http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF.pdf>

audience. A survey of literature on multiagent planners together with the competition poll provided enough information to set the rules for the competition so that an ample amount of already existing multiagent planners could compete and still the key motivations of the competition remained satisfied.

The fundamental discriminator of current multiagent planners is whether they can work distributively on multiple interconnected physical machines, or not. To accommodate planners running in either modes, the competition was split in two tracks (see Figure 1):

- *Centralized Track*, aiming for maximal compatibility with classical IPC and existing multiagent planners; the input of a planner was either unfactored or factored MA-PDDL; the planners run on a single machine, with no other restrictions or requirements; communication was not restricted; privacy was not enforced.
- *Distributed Track*, much more strict aiming for a proper multiagent setting; the input was limited to distributed factored MA-PDDLs for each agent; planners run distributively on a grid of machines; planners had to communicate over TCP/IP; preservation of privacy of the local data was required.

**Evaluation and Benchmarks** Each run of a planner in the competition was restricted to 30 minutes on 4 computational cores and 8GB per machine.

The metrics used to compare the planners were coverage (number) of solved problems, IPC Score over the plan quality, and IPC score over the planning time. In the distributed track, the plan quality was evaluated both in terms of total cost (sum of costs of all used actions) and makespan (the maximum timestep of the plan if executed in parallel).

The planners were evaluated over a set of 12 benchmark domains. The domains were motivated by important and interesting real-world problems and/or by problems exposing and testing theoretical features of the planners. We used domains from literature on multiagent planning: BLOCKSWORLD, DEPOT, DRIVERLOG, ELEVATORS08, LOGISTICS00, ROVERS, SATELLITES, SOKOBAN, WOODWORKING, and ZENOTRAVEL, each with 20 problem instances, with varying size, number of objects, constants, agents, and thus complexity. Additionally, we have added two novel domains inspired by well-known multiagent problems, not modeled in MA-STRIPS or MA-PDDL previously: TAXI and WIRELESS. The first one was a model of on-demand transport by taxis in a city, while the other modeled a group of communicating autonomous nodes in a wireless sensor network.

The validity and quality of plans was evaluated using the VAL<sup>3</sup> tool, which can handle parallel plans and performs the mutex checks.

**Selected Results** For the centralized track, we have received 12 planners in 17 configurations prepared by 8 teams. For the distributed track 6 configurations of 3 planners by 3 teams. Complete, detailed, and interactive results including detailed description of the

---

<sup>3</sup><http://www.inf.kcl.ac.uk/research/groups/planning>









Coverage					
centralized track			distributed track		
1.-2.	ADP 	222	1.	PSM 	180
3.	MAP-LAPKT 	216	2.	MAPlan 	174
4.	CMAP 	210	3.	MH-FMAP 	107

Table 1: Best performing planners in the metrics of solved problems out of overall 240 benchmarks.

planners and their authors can be found on the official competition webpage<sup>4</sup>, selected results are presented in Table 1.

The winning planners of the centralized track were two variants of the ADP (Agent Decomposition-based Planner) planner based on the idea of automatic decomposition of classical planning problems to multiple agents, the MAP-LAPKT planner based on solving of an encoded multiagent problem by a classical planner and CMAP based on subgoal extraction and factored compilation to classical planning.

The distributed track won a variant of the PSM planner based on intersection of finite automata representing sets of agents’ local plans coined Planning State Machines (PSM). Second place occupied MAPlan, a distributed heuristic search planner. The multi-heuristic partial-order forward-chaining planner MH-FMAP placed as third.

## 2.5 Demonstrator

The demonstrator (RT5) is based on the Tactical Environment [5] previously developed within a series of CERDEC-funded projects Tactical AgentFly and Tactical AgentScout. The environment features a high-fidelity simulation with full 3D physics and visualization. We used this toolkit to create challenging and dynamic environment to test and demonstrate the developed multiagent (online) planning techniques.

The demonstration consists of a mission domain specification and application of two multiagent algorithms for sequential decision making to provide behavior for the agents representing the tactical units. The Static Enemy Scenario, uses the MADLA planner [9, 10] and in the Dynamic Enemy Scenario, the Linear Side Information (LSI) utilizing algorithm [8] is used for both allied and enemy units.

## Scenario Overview

The used scenario comprise of an urban environment (a small village) with three possible types of units—a convoy (transport unit), infantry (combat units), and guards (defense units represented by the AFV Stryker). The convoy and guards are wheeled, infantry move on foot. There is a set of waypoints which can be connected either by a road or by a

<sup>4</sup>CoDMap results: <http://agents.cz/codmap/results>



Figure 2: A 2D and 3D visualization of the simulation environment.

path. The wheeled units can move only on roads, while units moving on feet can use both roads and paths. Some of the waypoints are also connected by line-of-sight (road or path does not imply line-of-sight) among buildings. The goal is to move the convoy safely from an initial waypoint to another goal waypoint.

In Figure 2, we show two examples of 2D and 3D visualizations of the simulated environment (in this particular case with static enemies). In the 2D top-view, execution of a prepared mission plan has just been started. The infantry agent is approaching the enemy infantry from a blind angle in order to disarm it. The **convoy** and the defending AFV vehicle (denoted as a **guard**) are prepared to follow the plan once the area has been cleared. In the 3D screen-shot, we show the convoy vehicle, the AFV Stryker with visualized plan and the infantry unit in the distance (green person avatar). The visualization is based on the original Tactical Environment visualization.

### Mission Domain Specification

The mission domain (coined CONVOY) models the scenario in a classical planning language. Two simple examples of problems in the CONVOY domain are shown in Figure 3. The red locations/connections are roads, the blue locations/connections are paths and the dotted lines show line-of-sight. **C** denotes the convoy, **G** denotes the goal location, **g1** and **g2** denote the guards, **i1** and **i2** denote the infantry and **e1** and **e2** denote the enemy infantry units. The first example comprise three agents (**C**, **i1**, **g1**), the second five agents (**C**, **i1**, **i2**,

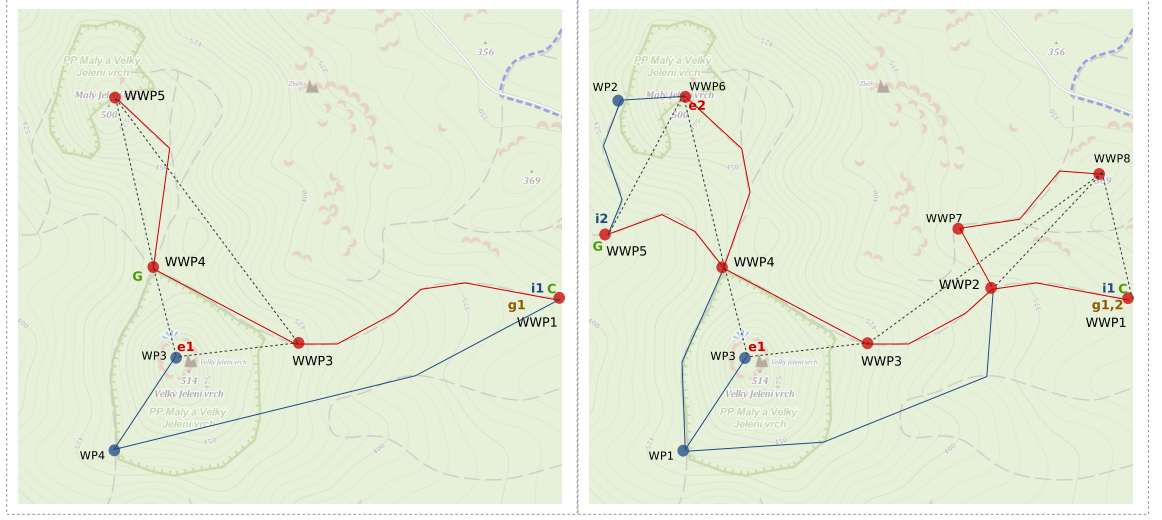


Figure 3: Two simple problems in the CONVOY demonstration domain (*convoy-a3* on left and *convoy-a5* on right).

g1, g2). In these particular cases, the enemies are static, hence not needed to be modeled as agents. As it will be shown later, in the scenario with dynamic enemies, both sides of the conflict are modeled equally, thus all units are represented by agents with antagonistic goals.

Figure 4 shows the listing of the domain definition in Planning Domain Description Language (PDDL) [7]. First of all, the unit types (*wunit* denotes a wheeled unit) are defined, then the connection predicates (*connected-w* is a road, *connected-f* is a path), line-of-sight, locations of units, activity of enemies, guarding/watching and danger predicates. After the predicates, the definition of actions follows. First three actions are move actions of the three distinct unit types. Notice that a convoy unit cannot move to a waypoint which is not guarded, the guard unit cannot move when it is guarding and both guard unit and infantry unit cannot move to a waypoint which is endangered by some active enemy. Next action is the *move-attack* action which moves an infantry unit to a location of an enemy unit and suppresses the enemy unit at the same time. Last two actions are used by the guard unit to guard and stop guarding (*unguard*) a specific waypoint.

A small CONVOY problem definition is listed in Figure 5. In the problem, there are six waypoints and one unit of each type. The initial state corresponds to the left illustration in Figure 3. Notice that both *wwp3* and *wwp4* are endangered by the enemy unit.

Solution to the *convoy-a3* problem is listed in Figure 6. According to the solution, the infantry first moves to suppress the enemy unit. Then the guard moves to a vantage point (*wwp6*) and finally the convoy is moved through guarded locations to the goal. Slightly more complex problem is depicted in the right illustration of Figure 3 (*convoy-a5*). The solution is listed in Figure 6. Similarly to the previous solution, the enemies are first

```

(define (domain convoy)
  (:requirements :strips :adl :typing)
  (:types
    wp wunit unit - object
    convoy guard - wunit
    infantry enemy wunit - unit
  )
  (:predicates (connected-f ?wp1 - wp ?wp2 - wp)
    (connected-w ?wp1 - wp ?wp2 - wp)
    (line-of-sight ?wp1 - wp ?wp2 - wp)
    (at ?u - unit ?wp - wp)
    (active ?e - enemy)
    (guarding ?g - guard)
    (danger ?wp - wp ?e - enemy)
    (guarded ?wp - wp ?g - guard)
  )
  (:action move-c
    :parameters (?c - convoy ?from - wp ?to - wp ?g - guard)
    :precondition (and (at ?c ?from) (connected-w ?from ?to) (guarded ?to ?g))
    :effect (and (at ?c ?to) (not (at ?c ?from)))
  )
  (:action move-g
    :parameters (?g - guard ?from - wp ?to - wp)
    :precondition (and (at ?g ?from) (connected-w ?from ?to) (not (guarding ?g))
      (forall (?e - enemy) (not (and (danger ?to ?e) (active ?e))))))
    :effect (and (at ?g ?to) (not (at ?g ?from)))
  )
  (:action move-i
    :parameters (?i - infantry ?from - wp ?to - wp)
    :precondition (and (at ?i ?from) (connected-f ?from ?to) (forall (?e - enemy)
      (not (and (danger ?to ?e) (active ?e))))))
    :effect (and (at ?i ?to) (not (at ?i ?from)))
  )
  (:action move-attack
    :parameters (?i - infantry ?from - wp ?to - wp ?e - enemy)
    :precondition (and (at ?i ?from) (at ?e ?to) (connected-f ?from ?to))
    :effect (and (at ?i ?to) (not (at ?i ?from)) (not (active ?e)))
  )
  (:action guard
    :parameters (?g - guard ?from - wp ?to - wp)
    :precondition (and (at ?g ?from) (line-of-sight ?from ?to) (not (guarding
      ?g)) (forall (?e - enemy) (not (and (danger ?to ?e) (active ?e))))))
    :effect (and (guarded ?to ?g) (guarding ?g))
  )
  (:action unguard
    :parameters (?g - guard ?wp - wp)
    :precondition (and (guarding ?g) (guarded ?wp ?g))
    :effect (and (not (guarded ?wp ?g)) (not (guarding ?g)))
  )
)

```

Figure 4: Definition in PDDL for the CONVOY domain.

```

(define (problem convoy-1g-1i)
  (:domain convoy)
  (:requirements :adl :strips :typing)
  (:objects
    wwp1 - wp
    wwp3 - wp
    wwp4 - wp
    wwp5 - wp
    wp3 - wp
    wp4 - wp
    c - convoy
    g1 - guard
    i1 - infantry
    e1 - enemy)
  (:init
    (connected-w wwp1 wwp3) (connected-w wwp3 wwp1)
    (connected-w wwp3 wwp4) (connected-w wwp4 wwp3)
    (connected-w wwp4 wwp5) (connected-w wwp5 wwp4)

    (connected-f wwp1 wwp3) (connected-f wwp3 wwp1)
    (connected-f wwp3 wwp4) (connected-f wwp4 wwp3)
    (connected-f wwp4 wwp5) (connected-f wwp5 wwp4)
    (connected-f wwp1 wp4) (connected-f wp4 wwp1)
    (connected-f wp4 wp3) (connected-f wp3 wp4)

    (line-of-sight wp3 wwp3) (line-of-sight wwp3 wp3)
    (line-of-sight wp3 wwp4) (line-of-sight wwp4 wp3)
    (line-of-sight wwp5 wwp4) (line-of-sight wwp4 wwp5)
    (line-of-sight wwp5 wwp3) (line-of-sight wwp3 wwp5)

    (at c wwp1)
    (at g1 wwp1)
    (at i1 wwp1)
    (at e1 wp3)

    (active e1)

    (danger wwp3 e1)
    (danger wwp4 e1))
  (:goal
    (and (at c wwp4)))
)

```

Figure 5: Problem definition in PDDL of a CONVOY problem.

```

0: move-i i1 wwp1 wp4
1: move-attack i1 wp4 wp3 e1
2: move-g g1 wwp1 wwp3
3: move-g g1 wwp3 wwp4
4: move-g g1 wwp4 wwp5
5: guard g1 wwp5 wwp3
6: move-c c wwp1 wwp3 g1
7: unguard g1 wwp3
8: guard g1 wwp5 wwp4
9: move-c c wwp3 wwp4 g1

```

(a)

```

0: move-i i1 wwp1 wwp2
1: move-i i1 wwp2 wp1
2: move-g g2 wwp1 wwp2
3: move-i i2 wwp5 wp2
4: move-attack i1 wp1 wp3 e1
5: move-attack i2 wp2 wwp6 e2
6: move-g g2 wwp2 wwp3
7: move-g g2 wwp3 wwp4
8: move-g g1 wwp1 wwp2
9: move-g g1 wwp2 wwp7
10: move-g g1 wwp7 wwp8
11: move-g g2 wwp4 wwp6
12: guard g1 wwp8 wwp2
13: move-c c wwp1 wwp2 g1
14: unguard g1 wwp2
15: guard g1 wwp8 wwp3
16: move-c c wwp2 wwp3 g1
17: guard g2 wwp6 wwp4
18: move-c c wwp3 wwp4 g2
19: unguard g2 wwp4
20: guard g2 wwp6 wwp5
21: move-c c wwp4 wwp5 g2

```

(b)

Figure 6: Solution to the *convoy-a3* (a) and *convoy-a5* (b) problems.

suppressed, then the guards are positioned on two vantage points (wwp4 and wwp8) and finally the convoy safely moves to the goal position.

**Tactical Planning** Recall that the tactical simulation of demonstrator is based on the Tactical Environment [5]. The state of the environment is represented by sets of state variable containers called *state storages*. Each state storage is responsible for holding a specific part of the current state, i.e., all state storages together constitute the full description of the current state of the simulated environment. All dynamic objects in the environment are denoted as *simulation entities*. A simulation entity is a simulated embodiment with a related controlling agent. The environment model is accompanied by a functional part, describing the behavior of the simulation. To ensure properties of the tested algorithms during the implementation process, the simulation platform facilitates construction of experiment suites allowing execution of reproducible experiments.

The Tactical Environment proposed in [5] was enriched with a new type of unit—a Domain-Independent Multiagent Planning (DIMAP) Physical Agent. Such unit represents any kind of agent which has the capability to participate on the multiagent planning process and which has a physical representation eventually executing the plan. The agent’s

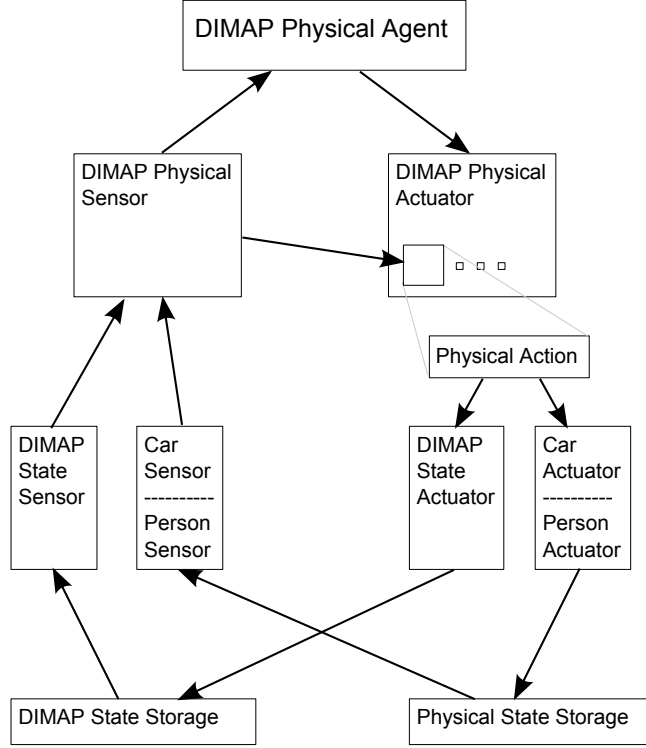


Figure 7: Architecture of an agent for (online) multiagent planning/decision making in the tactical environment of the demonstrator. The architecture for the static and dynamic enemy scenarios shares the same structure, the only (implementation) difference is in naming of the simulation objects (DIMAP Physical *Game* Agents, *Game* Sensors, *Game* Actuators, etc. for the units in the scenario with dynamic enemies).

architecture is depicted in Figure 7. According to a common agent architecture of the Tactical Package, the agent incorporates several sensors and actuators in a hierarchical manner, with the lowest layer being a storage of the world state.

Let us examine the architecture from bottom up. In the storage layer, there are two storage instances, shared by all agents. The Physical State Storage is similar to the classical Tactical Package storage and contains the state of the simulation world, i.e., positions and orientations of the objects. The DIMAP State Storage has been added to store the symbolic representation of the world used by planning. Currently we use a Java-based planner and therefore we were able to re-use the actual data structures—the DIMAP State Storage contains a DIMAP State, which is then modified by applied DIMAP actions.

The first sensor/actuator layer comprises of two sensor/actuator pairs, each affecting

one of the storage instances. In fact, there is a set of physical sensors and actuators, but in this level of abstraction we can safely treat them as a single sensor and single actuator. The physical actuators perform the actions in the simulation world, be it a simple action such as turning of an enemy towards a waypoint it endangers, or a complex action such as planning a path using trajectory planner and executing it. The physical sensor then reports changes in the world, most notably that the movement was finished. The DIMAP sensor/actuator pair is much simpler—the actuator applies given DIMAP Action to the current DIMAP State and the sensor reports that the symbolic world state has changed to all agents.

The top sensor/actuator layer contains a single sensor and actuator, both combining the symbolic and physical views of the world. In the case of the sensor, the combination is trivial—the sensor aggregates the DIMAP sensor and all the physical sensors. The actuator is slightly more intricate—as we intend to keep the architecture domain-independent, we needed to represent various actions present in the current domain. This is achieved by maintaining a collection of Physical Action instances, each representing a type of DIMAP action (determined by a label, corresponding to the PDDL operators), all having a common interface.

Each Physical Action uses the DIMAP State Actuator to modify the symbolic world state, according to the particular action instance (an operator, or action type, can have various parameters modifying how the action actually changes the state). In addition to that, some actions may have side effects in the physical world, which are reflected using the physical actuators. In some situations, the application of the action’s side effects are not as trivial. Take for instance a simple movement action *move-c-A-B* which should move the convoy *c* from location *A* to location *B*. When the physical aspect of the action is applied, it means that a trajectory is planned (may take some time) and then the convoy follows the trajectory in order to reach the destination *B* (takes some time). If we applied the symbolic effect of the action (which is something like *at-c-B*) immediately, the symbolic representation of the world would represent a world state, where the convoy is already at its destination, but the physical state of the world would be that the convoy is still planning its trajectory or, at best, is on its way. Therefore we need to apply the symbolic action effect only when the physical part of the action is finished. In order to do so, the Physical Action has access to the DIMAP Physical Sensor and may register callbacks for events such as that the agent has reached end of its plan.

At the topmost layer there is the DIMAP Physical Agent itself. The agent incorporates the decision making algorithm (the MADLA planner and the LSI algorithm in the Static and Dynamic Enemy Scenarios respectively), the communication interface and the DIMAP Physical sensor/actuator pair. The particular decision making process differs according to the used algorithm. The details are presented in the following two subsections.

Whenever the global symbolic DIMAP state is changed, which is detected by the sensors in the lower layers, or initially when the initial decision is made, each agent checks its decided action against the current DIMAP state. If all preconditions are met, the action is executed by the actuators in the lower layers.



## Static Enemy Scenario

In this scenario, there are fixed enemy units endangering from cover some of the waypoints (we assume the AFV cannot attack them frontally) and an allied infantry squad which is able to disarm the enemy units by ambushing them (moving to their position from the opposite direction they watch), thus freeing the passage.

The safety of the convoy is granted by a guard watching a waypoint to which the convoy is moving (the convoy cannot move to a waypoint which is not guarded). A single guard can guard a single waypoint which must be in line-of-sight from the position of the guard. When the guard is watching a waypoint, it cannot move. A guard can move only to a position which is not endangered by an enemy.

We use the initial state of the simulated environment to generate a formal multi-agent planning problem in form of files in PDDL which is then solved by the distributed multi-agent planner MADLA developed within RT3. The resulting multiagent plan is then executed in the simulated environment by the transport vehicle, AFV and the infantry squad. The vehicles utilize a maneuver-based trajectory planner, which is a part of the Tactical Environment package, to plan trajectories between the waypoints. Other actions (e.g., defense of a waypoint or disarming an enemy) are represented visually.

In order to use the MADLA Planner, a symbolic representation of the simulated world has to be generated. The symbolic representation used by the MADLA Planner as input has the form of Multi-Valued Planning Task (MPT) [3]. MPT consists of a finite set of variables, each with associated finite domain of possible values. A partial state is an assignment to some of the variables, a state is an assignment to all the variables. The problem description consists of an initial state, a goal partial state and a set of actions. Each action consist of a precondition and an effect. The precondition is an assignment to some of the variables (in fact a partial state) which must be true in a state in order to apply the action, the effect is again a partial assignment (state) which describes new values of variables after the action is applied.

As the problem at hand is multiagent, the problem description is factored for the separate agents. The factorization follows the MA-STRIPS [2] definition—a variable-value pair is public, if it is shared among actions of two (or more) distinct agents, otherwise it is private. An action is public, if it uses some of the public variable-value pairs. A public action can use both public and private variable-value pairs, such public action is seen by other agents as a projection—it is stripped-off all private preconditions and effects.

Let  $\mathcal{W}$  be the set of all waypoints in the map (currently all waypoints are reachable by all agents). Each of the agents is represented by the following variables, values and actions:

**Convoy** Position of a convoy  $c$  is represented by a single variable  $c-at$  with values being all waypoints in the map, i.e.,  $c-at \in \mathcal{W}$ . Target location of the convoy is part of the goal state. For each convoy, each guard and each two waypoints, there is an action *move- $c$ -from- $to$ - $g$*  which represents the movement of convoy between the two locations, while the *to* waypoint is guarded by the guard  $g$ . Precondition of the action are that the convoy is at waypoint *from* and the waypoint *to* is guarded by guard  $g$

(*guarded-by-g* variable). The effect is that the convoy is at *to* and that the guarded waypoint has been reached (see guard). The *move* action is public, because it shares the *guarded-by-g* variable with the guard's *guard/unguard* actions.

**Guard** Similarly to the convoy, the position of a guard *g* is represented by variable *g-at*  $\in \mathcal{W}$ . In addition, there is a variable *guarded-by-g*  $\in \mathcal{W} \cup \{\perp\}$  stating which waypoint is currently guarded by the guard *g* ( $\perp$  represents that no waypoint is guarded), and a *guarded-wp-g*  $\in \{REACHED, UNREACHED\}$  variable, stating, whether the currently guarded waypoint has been already reached, or not. This is necessary in order to synchronize the actions. A guard has a simple *move-g-from-to* action which is private, and a *moved-g-from-to* action, which is used in the situation, when the *to* waypoint is in danger of some enemy (see the enemy description), and has additional precondition, that such enemy must be disarmed. This action is public because it interacts with the infantry actions. Another public actions are the *guard-g-from-to* actions. The *guard* actions are generated for each two waypoints within a predefined range and with direct line-of-sight visibility between them based on the current environment model. The action *guard-g-from-to* has precondition  $\{g-at=from, guarded-by-g=\perp\}$  and effect  $\{guarded-by-g=to, guarded-wp-g=UNREACHED\}$ . In addition, there is again a variant of the action, named *guarddd*, which is used for the endangered waypoints and has an additional precondition similarly to the *moved* action. Finally, there is one private *unguard-g* action per guard, which if *guarded-wp-g*=*REACHED* changes the value of *guarded-by-g* to  $\perp$ .

**Infantry** As usual, an infantry *i* has one variable *i-at*  $\in \mathcal{W}$  describing its current location and three actions. First is simple private *move-i-from-to* which changes the value of *i-at*, second is the public *moved* variant with additional precondition  $\{to-danger-by-e=F\}$  for all enemies endangering the waypoint *wp*. Last action is specific to the infantry agents, it is a *moveattack-i-from-to-e* action, which is generated for each waypoint *to* occupied by some enemy and each waypoint *from* connected to it. The action preconditions are obvious, the infantry must be at *from* and the enemy must be at *to*. The effect of *moveattack* action is that all variables *wp-danger-by-e* are set to *F*, and that the infantry moves to the waypoint *to*. This action is public.

**Enemy** While an enemy is not an agent *per se*, there is a variable, *wp-danger-by-e*  $\in \{T, F\}$ , associated with a waypoint *wp* and enemy *e*, if the waypoint is endangered by the enemy (or the enemy is at the waypoint). This variable is initially set to true (*T*), and is public, because it is required in preconditions of guard actions and is changed by infantry actions.

The decision making process in the scenario with static enemy uses an off-line planning. First, the agents invoke the MADLA planner, in order to synthesize a plan in cooperation with other agents. When the goal state is reached by any of the agents, the plan reconstruction phase is initiated. In this phase, the agents reconstruct the plan backwards from the

goal state, starting from the agent which reached the goal. Since it is possible that multiple agents initiate the reconstruction process simultaneously, the agent’s id and the plan’s cost are included in the reconstruction messages. The least cost plan started by the agent with the lowest id is kept, other plans are discarded and their reconstruction is halted, when the redundancy is detected. Each agent stores the parts of the plan reconstructed by the particular agent.

Once the plan is reconstructed, each agent has a list of actions to be executed. Since the plan was constructed using a DIMAP planner, this implicit synchronization is enough to successfully execute the plan in an (deterministic) environment—no further communication is needed. This approach also allows to naturally parallelize actions where possible.

### Dynamic Enemy Scenario

In contrast to the previous scenario, in this case, the enemy units are not fixed and try to fulfill their own goals. Types of the units are the same for both sides of the conflict, however their numbers in the field can differ. The goal of both sides is to get their convoys to the goal positions. The initial position of the convoy of one side is the goal position of the enemy convoy and vice versa. A winning side of the conflict is that which gets its convoy to its goal position first. If no side can get its convoy to the goal position the result is a draw.

Each unit (regardless its type) can be in the Dynamic Enemy Scenario incapacitated by an attack from an infantry. The mechanics of the attack slightly differs from the previous scenario. The infantry attacks from its position and does not move during the attack. An incapacitated unit cannot act anymore (move, attack, guard, etc.). If two infantries attack each other in one time step, both are incapacitated. If a convoy is incapacitated its side cannot win by definition of the goal, but can act such that the conflict end as a draw. As the infantry attacks directly, there are no explicitly endangered waypoints as in the scenario with static enemies. Implicitly all waypoints the infantry can move to from its position are endangered.

A convoy no longer needs guarded waypoints it is moving to. An enemy unit cannot move to a guarded waypoint though. That means, a guard can obviate an infantry to move to vicinity of the convoy and thereby obviate the possibility of an attack on the convoy. The principle that a single guard can guard a single waypoint which must be in line-of-sight from its position is kept.

Additionally, a wheeled unit cannot move to a waypoint already occupied by another wheeled unit. This principle held implicitly in the scenario with the static enemies as the convoy could move only to a guarded waypoint and the AFV could not guard the same waypoint it was staying at. In this scenario, we use a variable *empty-wp*  $\in \{\perp, \top\}$  indicating whether the waypoint *wp* is empty of wheeled units.

The requirement on uncertainty caused by need to counteract the adversaries required deployment of an online planner we developed in context of RT3, namely the LSI algorithm—Linear Side Information utilizing Monte-Carlo sampling. Since LSI was designed with mul-

tiagentness in mind, it is suitable for the simulated mission with more units on both sides of the conflict. The algorithm is, however, exponential with increased lookahead depth which is proportional to the scale of the problem. As the number of possible combinations of actions applicable in each step grows exponentially with the number of agents, the algorithm has to use an additional assumption to provide decisions for all the agents in tractable time. LSI achieve this by assuming the actions of particular agents can be combined without any influence on each other. Such assumption works well in cases the units are separated or work with different resources. Provided that the units have to cooperate, or to resolve conflicts in their behavior, such assumption can be rather misleading, however still it is the state-of-the-art approach.

LSI is a selection algorithm for combinations of actions of the agents minimizing estimation of simple (simulated) regret after executing the combination and random acting of both sides till one of the states is reached (Monte Carlo sampling). In practice, that means the algorithm selects actions for all agents for a current state of the environment such that simulated rewards after a limited number of samples is maximal. In a game-like scenario we described here, the reward of the final states can be simply 1 for win, 0 for tie, and -1 for loss, however simulating always all samples to a final state increases the complexity and in case of cycles in the game can never terminate. Therefore LSI uses a heuristic evaluation function which after a predefined number of simulation steps evaluates the reached state and uses the evaluation as the reward. In our particular case, the heuristic evaluates distance of the convoy to the goal waypoint (interval 0–10000) and sums it with predefined values of active units (100 for infantry, 1000 for a guard, and 10000 for a convoy).

As the decision of the two sides of the conflict are not coordinated (the agents are behaving competitively), actions can get into conflicts. Although the selected action combinations by one side has to be non-conflicting, we have to resolve conflicts during execution between the enemies. At each step of the DIMAP State model, conflicting actions are replaced by empty actions, thus the conflict cannot happen during the following execution. Two actions are in a conflict if their effects change the same variable in the MPT description, e.g., if two wheeled units decide to move to the same waypoint, the *empty-wp* variable would be set by both units to *empty-wp* =  $\perp$ . Such behavior is detected as a conflict and both move actions would be replaced by empty actions, i.e., the units will not move in the particular step.

The process of Monte Carlo sampling uses the STRIPS applicability of actions described in the CONVOY domain, therefore the action model used is the same as in the static case, but with different decision making technique. To increase efficiency of picking a random applicable action during the simulation, we have implemented a action cache using the position of a unit as the key. To use the CONVOY domain for a conflict scenario, we had to enrich and adjust the models of the unit types:

**Convoy** Each convoy  $c$  can be *alive*- $c \in \{\perp, \top\}$ . Position of a convoy  $c$  is represented by a single variable  $c-at$  with values being all waypoints in the map, i.e.,  $c-at \in \mathcal{W}$ . Target location of the convoy is part of the goal state. Each team has its own set of

goals. For each convoy and each two waypoints, there is an action *move-c-from-to* which represents the movement of convoy between the two locations. Precondition of the action are that the convoy is *alive-c* =  $\top$ , at waypoint *from*, the waypoint *to* is empty of wheeled units (*empty-to* =  $\top$ ), and the waypoint *to* is not guarded by (see *guarded-wp* in next paragraph) an enemy guard. The effect is that the convoy is at *to* and that the *to* waypoint is not empty any more (*empty-to* =  $\perp$ ).

**Guard** Similarly to the convoy, a guard *g* can be *alive-g*  $\in \{\perp, \top\}$  and the position is represented by variable *g-at*  $\in \mathcal{W}$ . In addition, there is a variable *guarding-g*  $\in \mathcal{W} \cup \{\perp\}$  stating which waypoint is currently guarded by the guard *g* ( $\perp$  represents that no waypoint is guarded). Moreover, each waypoint *wp* is marked by *guarded-wp* as unguarded ( $\perp, \perp$ ), guarded by one side ( $\top, \perp$ ) or ( $\perp, \top$ ), or guarded by both sides of the conflict ( $\top, \top$ ). The information what AFV guards the waypoint is already in *guarding-g*, therefore it is not kept in the *guarded-wp* variable. A guard has a *move-g-from-to* action which behaves similarly as the move action of a convoy. Another actions are the *guard-g-from-to* actions. The *guard* actions are generated for each two waypoints within a predefined range and with direct line-of-sight visibility between them based on the current environment model. The action *guard-g-from-to* has precondition  $\{alive-g = \top, g-at = from, guarding-g = \perp, \dots\}$  and effect  $\{guarding-g = to, \dots\}$ . The actions are in two variants for *guarded-to* = ( $\perp, *$ ) and *guarded-to* = ( $\top, *$ ). There are also two variants of reverse actions *unguard-g-from-to* which sets *guarding-g* =  $\perp$  and reverts *guarded-to*. Additionally, guards can move only if *guarding-g* =  $\perp$ .

**Infantry** As usual, an infantry *i* can be *alive-i*  $\in \{\perp, \top\}$  and has one variable *i-at*  $\in \mathcal{W}$  describing its current location. Infantry can move by action *move-i-from-to* which changes the value of *i-at* and in contrast to convoy and guards, *empty-to* =  $\top$  is not required. The attack actions *attack-i-from-to-e* are generated for all combinations of waypoints *from* and *to* and enemies *e*. The infantry can attack only if it is at *from*, it is alive, the enemy unit is at *to* and it is also alive. The effect of attack actions is *alive-e* =  $\perp$ .

The decision making process in the scenario with dynamic enemies uses an on-line planning. The agents on both sides of the conflict invoke their instance of the LSI algorithm, in order to get actions they should execute. When the the actions are selected by LSI, the conflicts are pair-wise detected and prospectively resolved by the principle described in the previous paragraphs.

With the conflict free actions, the agents runs the lower levels of the DIMAP architecture using the selected actions by LSI. After low-level execution of the actions, and update in the DIMAP State Storage, LSI is run again with new DIMAP state and new actions are selected for next step and execution. This process is repeated until one of the final states is reached.



Figure 8: Selected initial states in the scenario with static enemies; small setting (left), large setting (right).

## Discussion and Conclusions

Although the MADLA planner and the online planning algorithm LSI were analyzed on commonly used benchmarks and the results were presented in their scientific articles, the research publications did not analyze deployment to high-fidelity simulation and environments with complex dynamics and low level actuators. As the common format of such publications is not suitable for presenting such practically oriented work, we have prepared a demonstrator featuring both algorithms in a suitable tactical scenario for which we used the Tactical Environment [5] previously developed within a series of CERDEC-funded projects Tactical AgentFly and Tactical AgentScout.

First, we have designed and implemented a common model of the tactical mission using STRIPS and PDDL languages. The mission domain specification outlined the model mechanics and defined types of units and their allowed behavior. The specification was validated with a multiagent planner on an extended multiagent model of the problem described in MA-STRIPS.

Based on the experience gain in this prototypical implementation, we have extended the both the domain description and the integration of the off-line multiagent planner. Additionally, we have created a simulation of the tactical environment and integrated the high-level STRIPS model with low-level control of units in the simulated tactical environ-

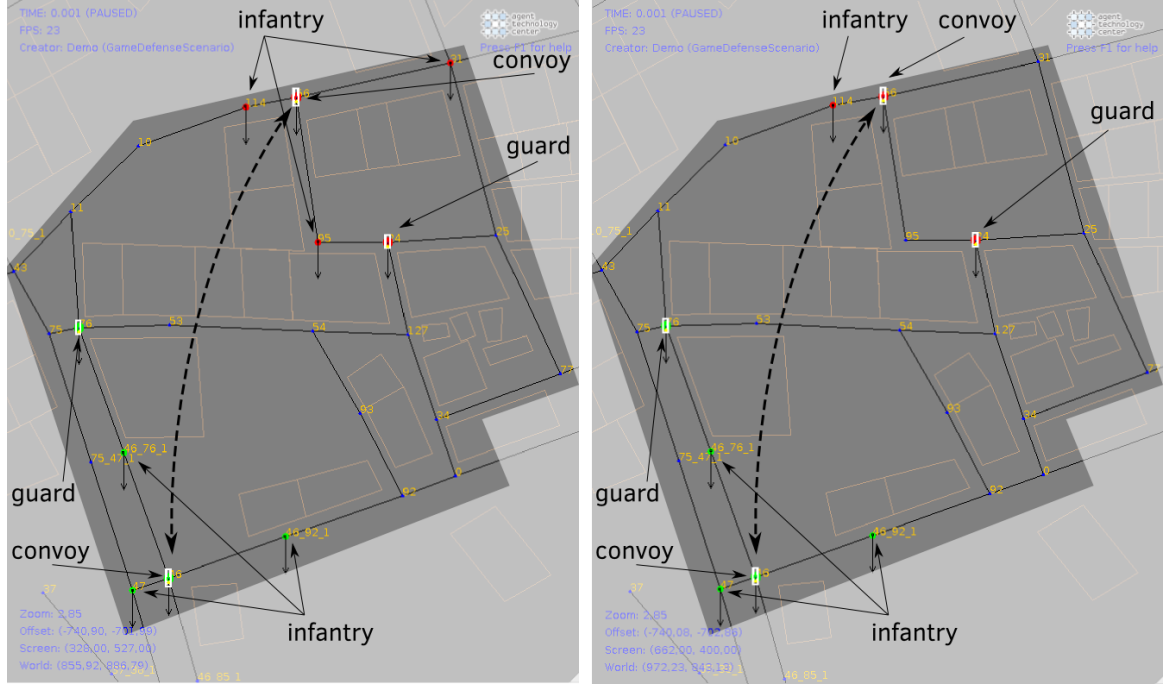


Figure 9: Selected initial states in the scenario with dynamic enemies; symmetric setting (left), asymmetric setting (right).

ment. To validate such integration, we built a generator of the planning problem instances for the off-line planner from the initial simulation states as depicted in Figure 8. The planner then in beforehand prepares a complete plan for the mission and the agents controlling the units synchronously execute the prescribed action ending in one of the goal states. As the multiagent planner is proven to always return a valid plan and the execution is in the static scenario always perfect, the agents always behave such that the goal is reached, provided that there exists a solution.

The assumption of a perfect action execution is valid only for a family of specialized problems. In a general case, the execution can fail. Theoretically, we have studied three types of planning under uncertainty. First, we have deepened our analysis from the previous CERDEC funded projects on repairing of multiagent plans in form of prefix and suffix reuse. Deployment of these techniques was published as part of [4]. Second, we have proposed a tractable compilation scheme for fault-tolerant planning with a limit on number of possible execution failures. Finally, we have proposed new algorithms in the family of Monte Carlo Tree Search and proposed a multiagent variant of Monte Carlo sampling for multiagent scenarios with actions executed in parallel.

The online multiagent planning algorithm was adapted to work with the STRIPS model of the actions in the model of a tactical mission and reusing and extending the principles

of integration of a off-line planner, we were able to extend the mission to complete conflict scenario with different types of units on both sides causing uncertainty in the execution of the planned actions. The uncertainty caused by a intelligent opponent are the “worst” which can happen, therefore the successful behavior of the agents against an opponent validates the principle and shows that the winning or losing ratio depends on numbers of the units. The initial stats of the conflict scenarios are depicted in Figure 9. The algorithm in each step of the simulation analyses the most prospective action of the agents which are then executed by the respective tactical units. The demonstrator show the behavior copies the positive result of the algorithm in the synthetic problems used in the theoretical analysis, thus validates applicability of the online planning under uncertainty in the simulated high-fidelity simulation of a tactical mission in the margins of the assumptions and model used.



## References

- [1] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 28–35, 2008.
- [2] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS*, pages 28–35, 2008.
- [3] M. Helmert. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.
- [4] A. Komenda. *Domain-independent Multiagent Plan Repair*. PhD thesis, Department of Computer Science and Engineering, Czech Technical University in Prague, 2013.
- [5] A. Komenda, J. Vokřínek, M. Čáp, and M. Pěchouček. Developing multiagent algorithms for tactical missions using simulation. *Intelligent Systems, IEEE*, 28(1):42–49, 2013.
- [6] D. L. Kovacs. A multi-agent extension of PDDL3.1. In *Proc. of the 3rd Workshop on the International Planning Competition (IPC)*, pages 19–27, 2012.
- [7] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.
- [8] A. Shleyfman, A. Komenda, and C. Domshlak. On Combinatorial Actions and CMABs with Linear Side Information. In *Proceedings of 21st ECAI*, pages 825–830, 2014.
- [9] M. Štolba and A. Komenda. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, pages 298–306, 2014.
- [10] M. Štolba and A. Komenda. MADLA: Planning with Distributed and Local Search. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 21–24, 2015.

## Appendix A. Results Access & File Structure

Access to the project repository:

url: <https://webdav.agents.fel.cvut.cz/data/projects/robust-planning/>

username: FA8655-12-1-2096

password: robustmultiagentplanning

File structure of the project repository:

**reports** contain all project reports

**papers** contain all research papers produces during the project

**videos** contain videos captured from the demonstrator

**tactical-env-demo** The video shows 2D and 3D visualization of the tactical environment used for the demonstrator together with all used unit types: the infantry, the guard (AFV Stryker) and the convoy.

**static-enemy** Demonstration of the small and large scenarios with the static enemy (see Figure 8).

**dynamic-enemy-asymmetric** A asymmetric conflict in the scenario with the dynamic enemy. One of the sides of the conflict has two more infantries which allows it to win both example runs.

**dynamic-enemy-symmetric** A symmetric variant of the previous scenario. In this case, the result is give much more by chance. The video shows three runs resulting as a tie, a win of one side and a win of the other side.

**dynamic-enemy-guards** The full scenario by the definition of the CONVOY domain with dynamic enemies in uncertain environment using the robust planning system we have developed. The scenario includes the guard AFVs. Explicit blocking of an enemy infantry by one of the guards is emphasized. The video contains two runs both ending as a tie.

**demo** contains the demonstrator

**build** The directory contains binaries of the demonstrator for JDK7.

**src** The directory contains Java sources of the demonstrator with dependencies.

To run the demonstrator, Java JRE7 has to be installed, at least 4GB RAM available and for 3D visualization a OpenGL 3.0 graphics card installed.

To build the demonstrator from sources Java JDK7 has to be installed. Use Maven (<https://maven.apache.org/>). For all but robust-planning-demo run `mvn install`, for robust-planning-demo run `mvn package`.

## **Appendix B. Selected Research Results**

# Simple Regret Optimization in Online Planning for Markov Decision Processes

**Zohar Feldman**

**Carmel Domshlak**

*Faculty of Industrial Engineering & Management,  
Technion - Israel Institute of Technology,  
Haifa, Israel*

ZOHARF@TX.TECHNION.AC.IL

DCARMEL@IE.TECHNION.AC.IL

## Abstract

We consider online planning in Markov decision processes (MDPs). In online planning, the agent focuses on its current state only, deliberates about the set of possible policies from that state onwards and, when interrupted, uses the outcome of that exploratory deliberation to choose what action to perform next. The performance of algorithms for online planning is assessed in terms of *simple regret*, which is the agent’s expected performance loss when the chosen action, rather than an optimal one, is followed.

To date, state-of-the-art algorithms for online planning in general MDPs are either best effort, or guarantee only polynomial-rate reduction of simple regret over time. Here we introduce a new Monte-Carlo tree search algorithm, BRUE, that guarantees *exponential-rate* reduction of simple regret and error probability. This algorithm is based on a simple yet non-standard state-space sampling scheme, MCTS2e, in which different parts of each sample are dedicated to different exploratory objectives. Our empirical evaluation shows that BRUE not only provides superior performance guarantees, but is also very effective in practice and favorably compares to state-of-the-art. We then extend BRUE with a variant of “learning by forgetting.” The resulting set of algorithms, BRUE( $\alpha$ ), generalizes BRUE, improves the exponential factor in the upper bound on its reduction rate, and exhibits even more attractive empirical performance.

## 1. Introduction

Markov decision processes (MDPs) are a standard model for planning under uncertainty (Puterman, 1994). An MDP  $\langle S, A, Tr, R \rangle$  is defined by a set of possible agent states  $S$ , a set of agent actions  $A$ , a stochastic transition function  $Tr : S \times A \times S \rightarrow [0, 1]$ , and a reward function  $R : S \times A \times S \rightarrow \mathbb{R}$ . Depending on the problem domain and the representation language, the description of the MDP can be either declarative or generative (or mixed). In any case, the description of the MDP is assumed to be concise. While declarative models provide the agents with greater algorithmic flexibility, generative models are more expressive, and both types of models allow for simulated execution of all feasible action sequences, from any state of the MDP. The current state of the agent is fully observable, and the objective of the agent is to act so to maximize its accumulated reward. In the finite horizon setting that will be used for most of the paper, the reward is accumulated over some predefined number of steps  $H$ .

The desire to handle MDPs with state spaces of size exponential in the size of the model description has led researchers to consider online planning in MDPs. In online planning,

the agent, rather than computing a quality policy for the entire MDP before taking any action, focuses only on what action to perform next. The decision process consists of a deliberation phase, aka planning, terminated either according to a predefined schedule or due to an external interrupt, and followed by a recommended action for the current state. Once that action is applied in the real environment, the decision process is repeated from the obtained state to select the next action and so on.

The quality of the action  $a$ , recommended for state  $s$  with  $H$  steps-to-go, is assessed in terms of the probability that  $a$  is sub-optimal, and in terms of the (closely related) measure of *simple regret*  $\Delta_H[s, a]$ . The latter captures the performance loss that results from taking  $a$  and then following an optimal policy  $\pi^*$  for the remaining  $H - 1$  steps, instead of following  $\pi^*$  from the beginning (Bubeck & Munos, 2010). That is,

$$\Delta_H[s, a] = Q_H(s, \pi^*(s, H)) - Q_H(s, a),$$

where

$$Q_H(s, a) = \mathbb{E}_{s'} [R(s, a, s') + Q_{H-1}(s', \pi^*(s', H - 1))].$$

With a few recent exceptions developed for declarative MDPs (Bonet & Geffner, 2012; Kolobov, Mausam, & Weld, 2012; Busoniu & Munos, 2012), most algorithms for online MDP planning constitute variants of what is called Monte-Carlo tree search (MCTS). One of the earliest and best-known MCTS algorithms for MDPs is the sparse sampling algorithm by Kearns, Mansour, and Ng (Kearns, Mansour, & Ng, 1999). Sparse sampling offers a near-optimal action selection in discounted MDPs by constructing a sampled lookahead tree in time exponential in discount factor and suboptimality bound, but independent of the state space size. However, if terminated before an action has proved to be near-optimal, sparse sampling offers no quality guarantees on its action selection. Thus it does not really fit the setup of online planning. Several later works introduced interruptible, anytime MCTS algorithms for MDPs, with UCT (Kocsis & Szepesvári, 2006) probably being the most widely used such algorithm these days. Anytime MCTS algorithms are designed to provide convergence to the best action if enough time is given for deliberation, as well as a gradual reduction of performance loss over the deliberation time (Sutton & Barto, 1998; Péret & Garcia, 2004; Kocsis & Szepesvári, 2006; Coquelin & Munos, 2007; Cazenave, 2009; Rosin, 2011; Tolpin & Shimony, 2012). While UCT and its successors have been devised specifically for MDPs, some of these algorithms are also successfully used in partially observable and adversarial settings (Gelly & Silver, 2011; Sturtevant, 2008; Bjarnason, Fern, & Tadepalli, 2009; Balla & Fern, 2009; Eyerich, Keller, & Helmert, 2010).

In general, the relative empirical attractiveness of the various MCTS planning algorithms depends on the specifics of the problem at hand and cannot usually be predicted ahead of time. When it comes to formal guarantees on the expected performance improvement over the planning time, very few of these algorithms provide such guarantees for general MDPs, and none breaks the barrier of the worst-case only *polynomial-rate reduction* of simple regret and choice-error probability over time.

This is precisely our contribution here. We introduce a new Monte-Carlo tree search algorithm, BRUE, that guarantees *exponential-rate* reduction of both simple regret and choice-error probability over time, for general MDPs over finite state spaces. The algorithm is based on a simple and efficiently implementable sampling scheme, MCTS2e, in which

---

**MCTS:** [input:  $\langle S, A, Tr, R \rangle$ ;  $s_0 \in S$ ]  
search tree  $\mathcal{T} \leftarrow$  root node  $s_0$   
**while** time permits:  
     $\rho \leftarrow \text{sample}(s_0, \mathcal{T})$   
     $\mathcal{T} \leftarrow \text{expand-tree}(\mathcal{T}, \rho)$   
    update-statistics( $\mathcal{T}, \rho$ )  
**return** recommend-action( $s_0, \mathcal{T}$ )

---

Figure 1: High-level scheme for regular Monte-Carlo tree sampling.

different parts of each sample are dedicated to different competing exploratory objectives. The motivation for this objective decoupling came from a recently growing understanding that the current MCTS algorithms for MDPs do not optimize the reduction of simple regret directly, but only via optimizing what is called cumulative regret, a performance measure suitable for the (very different) setting of reinforcement learning (Bubeck & Munos, 2010; Busoniu & Munos, 2012; Tolpin & Shimony, 2012; Feldman & Domshlak, 2012). Our empirical evaluation on some standard MDP benchmarks for comparison between MCTS planning algorithms shows that BRUE not only provides superior performance guarantees, but is also very effective in practice and favorably compares to state of the art. We then extend BRUE with a variant of “learning by forgetting.” The resulting family of algorithms, BRUE( $\alpha$ ), generalizes BRUE, improves the exponential factor in the upper bound on its reduction rate, and exhibits even more attractive empirical performance.

## 2. Monte-Carlo Planning

MCTS, a high-level scheme for Monte-Carlo tree search that gives rise to various specific algorithms for online MDP planning, is depicted in Figure 1. Starting with the current state  $s_0$ , MCTS performs an iterative construction of a tree  $\mathcal{T}$  rooted at  $s_0$ . At each iteration, MCTS issues a state-space sample from  $s_0$ , expands the tree  $\mathcal{T}$  using the outcome of that sample, and updates information stored at the nodes of  $\mathcal{T}$ . Once the simulation phase is over, MCTS uses the information collected at the nodes of  $\mathcal{T}$  to recommend an action to perform in  $s_0$ . For compatibility of the notation with prior literature, in what follows we refer to the tree nodes via the states associated with these nodes. Note that, due to the Markovian nature of MDPs, it is unreasonable to distinguish between nodes associated with the same state at the same depth. Hence, the actual graph constructed by most instances of MCTS forms a DAG over nodes  $(s, h) \in S \times \{0, 1, \dots, H\}$ . By  $A(s) \subseteq A$  in what follows, we refer to the subset of actions applicable in state  $s$ .

Numerous concrete instances of MCTS have been proposed, with UCT (Kocsis & Szepesvári, 2006) probably being the most popular such algorithm these days (Gelly & Silver, 2011; Sturtevant, 2008; Bjarnason et al., 2009; Balla & Fern, 2009; Eyerich et al., 2010; Keller & Eyerich, 2012a). To give a concrete sense of MCTS’s components, as well as to ground some intuitions discussed later on, below we describe the specific setting of MCTS corresponding to the core UCT algorithm, and Figure 2 illustrates the UCT tree construction, with  $n$  denoting the number of state-space samples.

- **sample:** The samples  $\rho = \langle s_0, a_1, s_1, \dots, a_k, s_k \rangle$  are all issued from the root node  $s_0$ . The sample ends either when a sink state is reached, that is,  $A(s_k) = \emptyset$ , or when  $k = H$ . Each node/action pair  $(s, a)$  is associated with a counter  $n(s, a)$  and a value accumulator  $\hat{Q}(s, a)$ . Both  $n(s, a)$  and  $\hat{Q}(s, a)$  are initialized to 0, and then updated by the **update-statistics** procedure. Given  $s_i$ , the next-on-the-sample action  $a_{i+1}$  is selected according to the deterministic UCB1 policy (Auer, Cesa-Bianchi, & Fischer, 2002a), originally proposed for optimal cumulative regret minimization in stochastic multi-armed bandit (MAB) problems (Robbins, 1952): If  $n(s_i, a) > 0$  for all  $a \in A(s_i)$ , then

$$a_{i+1} = \operatorname{argmax}_a \left[ \hat{Q}(s_i, a) + c \sqrt{\frac{\log n(s_i)}{n(s_i, a)}} \right], \quad (1)$$

where  $n(s) = \sum_a n(s, a)$ . Otherwise,  $a_{i+1}$  is selected uniformly at random from the still unexplored actions  $\{a \in A(s_i) \mid n(s_i, a) = 0\}$ . In both cases,  $s_{i+1}$  is then sampled according to the conditional probability  $\mathbb{P}(S|s_i, a_{i+1})$ , induced by the transition function  $Tr$ .

- **expand-tree:** Each state-space sample  $\rho = \langle s_0, a_1, s_1, \dots, a_k, s_k \rangle$  induces a state trace  $\langle s_0, s_1, \dots, s_i \rangle$  inside  $\mathcal{T}$ , as well as a state trace  $\langle s_{i+1}, \dots, s_k \rangle$  outside of  $\mathcal{T}$ . In principle,  $\mathcal{T}$  can be expanded with any prefix of  $\langle s_{i+1}, \dots, s_k \rangle$ ; a popular choice in prior work appears to be expanding  $\mathcal{T}$  with only the upper-most node  $s_{i+1}$ . (If  $\mathcal{T}$  is constructed as a DAG, it is expanded with the first node along  $\rho$  that leaves  $\mathcal{T}$ .)
- **update-statistics:** For each node  $s_i$  along  $\rho$  that is now part of the expanded tree  $\mathcal{T}$ , the counter  $n(s_i, a_{i+1})$  is incremented and the estimated  $Q$ -value is updated as

$$\hat{Q}(s_i, a_{i+1}) \leftarrow \hat{Q}(s_i, a_{i+1}) + \frac{R_i - \hat{Q}(s_i, a_{i+1})}{n(s_i, a_{i+1})}, \quad (2)$$

where  $R_i = \sum_{j=i}^{k-1} R(s_j, a_{j+1}, s_{j+1})$ .

- **recommend-action:** Interestingly, the action recommendation protocol of UCT was never properly specified, and different applications of UCT adopt different decision rules, including maximization of the estimated  $Q$ -value, of the augmented estimated  $Q$ -value as in Eq. 1, of the number of times the action was selected during the simulation, as well as randomized protocols based on the information collected at the root.

The key property of UCT is that its exploration of the search space is obtained by considering a hierarchy of forecasters, each minimizing its own *cumulative regret*, that is, the loss of the total reward incurred by exploring the environment (Auer et al., 2002a). Each such pseudo-agent forecaster corresponds to a state/steps-to-go pair  $(s, h)$ . In that respect, according to Theorem 6 of Kocsis and Szepesvári (2006), UCT asymptotically achieves the best possible (logarithmic) cumulative regret. However, as recently pointed out in numerous works (Bubeck & Munos, 2010; Busoniu & Munos, 2012; Tolpin & Shimony, 2012; Feldman & Domshlak, 2012), cumulative regret does not seem to be the right objective for online MDP planning, and this is because the rewards “collected” at the simulation

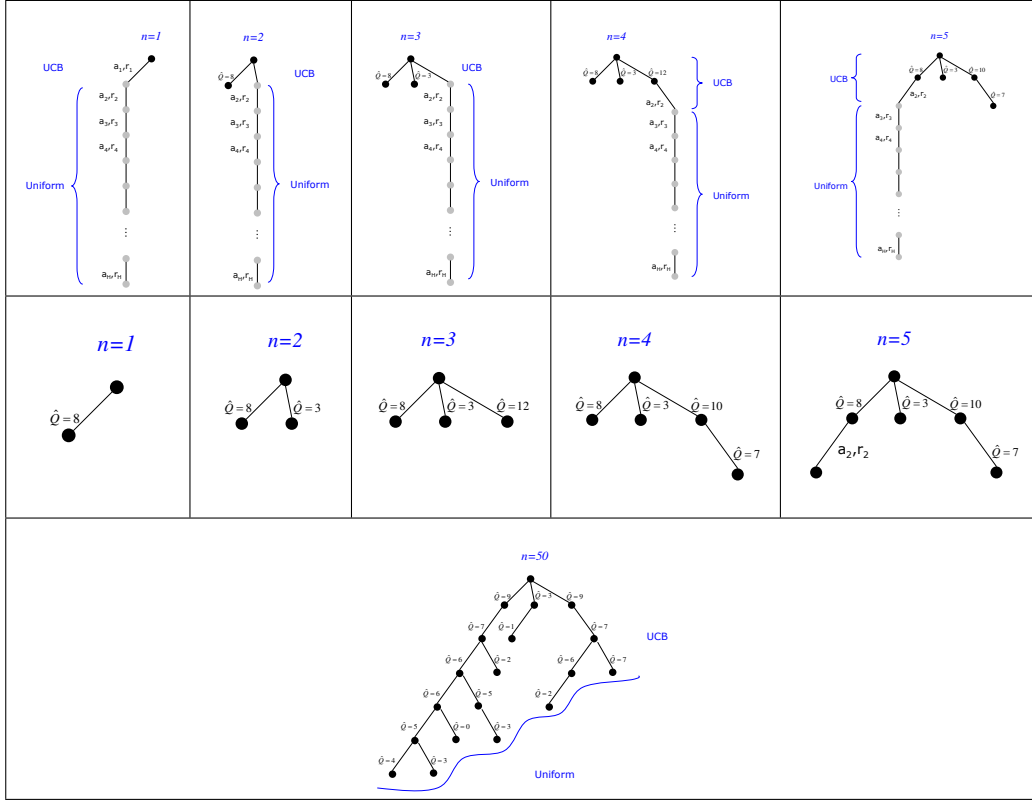


Figure 2: Illustration of the UCT dynamics

phase are fictitious. Furthermore, the work of Bubeck, Munos, and Stoltz (2011) on multi-armed bandits shows that minimizing cumulative regret and minimizing simple regret are somewhat competing objectives. Indeed, the same Theorem 6 of Kocsis and Szepesvári (2006) claims only a polynomial-rate reduction of the probability of choosing a non-optimal action, and the results of Bubeck et al. (2011) on simple regret minimization in MABs with stochastic rewards imply that UCT achieves only polynomial-rate reduction of the simple regret over time. Some attempts have recently been made to adapt UCT, and MCTS-based planning in general, to optimizing simple regret in online MDP planning directly, and some of these attempts were empirically rather successful (Tolpin & Shimony, 2012; Hay, Shimony, Tolpin, & Russell, 2012). However, to the best of our knowledge, none of them breaks UCT’s barrier of the worst-case polynomial-rate reduction of simple regret over time.

### 3. Simple Regret Minimization in MDPs

We now show that exponential-rate reduction of simple regret in online MDP planning is achievable. To do so, we first motivate and introduce a family of MCTS algorithms with a two-phase scheme for generating state space samples, and then describe a concrete algorithm from this family, BRUE, that (1) guarantees that the probability of recommending a non-



optimal action asymptotically converges to zero at an exponential rate, and (2) achieves exponential-rate reduction of simple regret over time.

### 3.1 Exploratory concerns in online MDP planning

The work of Bubeck et al. (2011) on pure exploration in multi-armed bandit (MAB) problems was probably the first to stress that the minimal simple regret can increase as the bound on the cumulative regret is decreases. At a high level, Bubeck et al. (2011) show that efficient schemes for simple regret minimization in MAB should be as exploratory as possible, thus improving the expected quality of the recommendation issued at the end of the learning process. In particular, they showed that the simple round-robin sampling of MAB actions, followed by recommending the action with the highest empirical mean, yields exponential-rate reduction of simple regret, while the UCB1 strategy that balances between exploration and exploitation yields only polynomial-rate reduction of that measure. In that respect, the situation with MDPs is seemingly no different, and thus Monte-Carlo MDP planning should focus on exploration only. However, the answer to the question of what it means to be “as exploratory as possible” with MDPs is less straightforward than it is in the special case of MABs.

For an intuition as to why the “pure exploration dilemma” in MDPs is somewhat complicated, consider the state/steps-to-go pairs  $(s, h)$  as pseudo-agents, all acting on behalf of the root pseudo-agent  $(s_0, H)$  that aims at minimizing its own simple regret in a stochastic MAB induced by the applicable actions  $A(s_0)$ . Clearly, if an oracle would provide  $(s_0, H)$  with an optimal action  $\pi^*(s_0, H)$ , then no further deliberation would be needed until after the execution of  $\pi^*(s_0, H)$ . However, the task characteristics of  $(s_0, H)$  are an exception rather than a rule. Suppose that an oracle provides us with optimal actions for *all* pseudo-agents  $(s, h)$  *but*  $(s_0, H)$ . Despite the richness of this information,  $(s_0, H)$  in some sense remains as clueless as it was before: To choose between the actions in  $A(s_0)$ ,  $(s_0, H)$  needs, at the very least, some ordinal information about the expected value of these alternatives. Hence, when sampling the futures, each non-root pseudo-agent  $(s, h)$  should be devoted to *two* objectives:

- (1) identifying an optimal action  $\pi^*(s, h)$ , and
- (2) estimating the actual value of that action, because this information is needed by the predecessor(s) of  $(s, h)$  in  $\mathcal{T}$ .

Note that both these objectives are *exploratory*, yet the problem is that they are somewhat competing. In that respect, the choices made by UCT actually make sense: Each sample  $\rho$  issued by UCT at  $(s, h)$  is a priori devoted *both* to increasing the confidence in that some current candidate  $a^\dagger$  for  $\pi^*(s, h)$  is indeed  $\pi^*(s, h)$ , as well as to improving the estimate of  $Q_h(s, a^\dagger)$ , while as if assuming that  $\pi^*(s, h) = a^\dagger$ . However, while such an overloading of the samples is unavoidable in the “learning while acting” setup of reinforcement learning, this should not necessarily be the case in online planning. Moreover, this sample overloading in UCT comes with a high price: As it was shown by Coquelin and Munos (2007), the number of samples after which the bounds of UCT on both simple and cumulative regret become meaningful might be as high as hyper-exponential in  $H$ .

### 3.2 Separation of Concerns at the Extreme

Separating the two aforementioned exploratory concerns is at the focus of our investigation here. Let  $s_0$  be a state of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$ ,  $K$  applicable actions at each state,  $B$  possible outcome states for each action, and finite horizon  $H$ . First, to get a sense of what separation of exploratory concerns in online planning can buy us, we begin with a MAB perspective on MDPs, with each arm in the MAB corresponding to a “flat” policy of acting for  $H$  steps starting from the current state  $s_0$ . A “flat” policy  $\pi$  is a minimal partial mapping from state/steps-to-go pairs to actions that fully specifies an acting strategy in the MDP for  $H$  steps, starting at  $s_0$ . Sampling such an arm  $\pi$  is straightforward as  $\pi$  prescribes precisely which action should be applied at every state that can possibly be encountered along the execution of  $\pi$ . The reward of such an arm  $\pi$  is stochastic, with support  $[0, H]$ , and the number of arms in this schematic MAB is  $K' = K^{\sum_{i=0}^{H-1} B^i} \approx K^{B^H}$ .

Now, consider a simple algorithm, **NaiveUniform**, which systematically samples each “flat” policy in a loop, and updates the estimation of the corresponding arm with the obtained reward. If stopped at iteration  $n$ , the algorithm recommends  $\pi(s_0)$ , where  $\pi$  is the arm/policy with best empirical value  $\hat{\mu}_{\pi,n}$ . By the iteration  $n$  of this algorithm, each arm will be sampled at least  $\lfloor \frac{n}{K^{B^H}} \rfloor$  times. Therefore, using the Hoeffding’s inequality, the probability that the chosen arm  $\pi$  is sub-optimal in our MAB is bounded by

$$\mathbb{P}\{\hat{\mu}_{\pi,n} > \hat{\mu}_{\pi^*,n}\} = \mathbb{P}\{\hat{\mu}_{\pi,n} - \hat{\mu}_{\pi^*,n} - (-\Delta_\pi) \geq \Delta_\pi\} \leq \exp\left(-\frac{\lfloor \frac{n}{K^{B^H}} \rfloor \Delta_\pi^2}{2H^2}\right), \quad (3)$$

where  $\Delta_\pi = \mu_{\pi^*} - \mu_\pi$ , and thus the expected simple regret can be bounded as

$$\mathbb{E}r_n \leq HK^{B^H} \exp\left(-\frac{\lfloor \frac{n}{K^{B^H}} \rfloor d^2}{2H^2}\right). \quad (4)$$

Note that **NaiveUniform** uses each sample  $\rho = (s_0, a_0, s_1, a_1, \dots, a_{H-1}, s_H)$  to update the estimation of only a single policy  $\pi$ . However, recalling that arms in our MAB problem are actually compound policies, the same sample can in principle be used to update the estimates of all policies  $\pi'$  that are consistent with  $\rho$  in the sense that, for  $0 \leq i \leq H-1$ ,  $\pi'(s_i, H-i)$  is defined and it is defined as  $\pi'(s_i, H-i) = a_i$ . The resulting algorithm, **CraftyUniform**, generates samples by choosing the actions along them uniformly at random, and uses the outcome of each sample to update all the policies consistent with it. Note that sampling the arms in **CraftyUniform** cannot be done systematically as in **NaiveUniform** because the set of policies updated at each iteration is stochastic.

Since the sampling is uniform, the probability of any policy to be updated by the sample issued at any iteration of **CraftyUniform** is  $\frac{1}{K^H}$ . For an arm  $\pi'$ , let  $N_{\pi',n}$  denote the number of samples issued at the  $n$  iterations of **CraftyUniform** that are consistent with the policy  $\pi'$ . The probability that  $\pi$ , the best empirical arm after  $n$  iterations, is sub-optimal is bounded by

$$\mathbb{P}\{\hat{\mu}_{\pi,n} > \hat{\mu}_{\pi^*,n}\} \leq \mathbb{P}\left\{\hat{\mu}_{\pi,n} - \mu_\pi \geq \frac{\Delta_\pi}{2}\right\} + \mathbb{P}\left\{\hat{\mu}_{\pi^*,n} - \mu_{\pi^*} \geq \frac{\Delta_\pi}{2}\right\}. \quad (5)$$

Each of the two terms on the right-hand side can be bounded as:

$$\begin{aligned}
\mathbb{P}\left\{\hat{\mu}_{\pi,n} - \mu_{\pi} \geq \frac{\Delta_{\pi}}{2}\right\} &\leq \mathbb{P}\left\{N_{\pi,n} \leq \frac{n}{2K^H}\right\} + \mathbb{P}\left\{N_{\pi,n} > \frac{n}{2K^H}, \hat{\mu}_{\pi,n} - \mu_{\pi} \geq \frac{\Delta_{\pi}}{2}\right\} \\
&\stackrel{(\dagger)}{\leq} e^{-\frac{n}{2K^{2H}}} + \sum_{i=\frac{n}{2K^H}+1}^n \mathbb{P}\{N_{\pi,n} = i\} \mathbb{P}\left\{\hat{\mu}_{\pi,n} - \mu_{\pi} \geq \frac{\Delta_{\pi}}{2} \mid N_{\pi,n} = i\right\} \\
&\leq e^{-\frac{n}{2K^{2H}}} + \mathbb{P}\left\{\hat{\mu}_{\pi,n} - \mu_{\pi} \geq \frac{\Delta_{\pi}}{2} \mid N_{\pi,n} = \frac{n}{2K^H} + 1\right\} \sum_{i=\frac{n}{2K^H}+1}^n \mathbb{P}\{N_{\pi,n} = i\} \\
&\leq e^{-\frac{n}{2K^{2H}}} + \mathbb{P}\left\{\hat{\mu}_{\pi,n} - \mu_{\pi} \geq \frac{\Delta_{\pi}}{2} \mid N_{\pi,n} = \frac{n}{2K^H} + 1\right\} \\
&\stackrel{(\ddagger)}{\leq} e^{-\frac{n}{2K^{2H}}} + e^{-\frac{n\Delta_{\pi}^2}{4K^H H^2}} \\
&\leq 2e^{-\frac{n\Delta_{\pi}^2}{4K^{2H} H^2}},
\end{aligned} \tag{6}$$

where  $(\dagger)$  and  $(\ddagger)$  are by the Hoeffding inequality. In turn, similarly to Eq. 4, the simple regret for **CraftyUniform** is bounded by

$$\mathbb{E}r_n \leq 4HK^{B^H} e^{-\frac{nd^2}{4K^{2H} H^2}}. \tag{7}$$

Since  $H$  is a trivial upper-bound on  $\mathbb{E}r_n$ , the bound in Eq. 7 becomes effective only when  $4K^{B^H} \exp\left(-\frac{nd^2}{4K^{2H} H^2}\right) < 1$ , that is, for

$$n > (K^2 B)^H \cdot 4 \left(\frac{H}{d}\right)^2 \log K. \tag{8}$$

Note that this transition period length is still much better than that of UCT, which is hyper-exponential in  $H$ . Moreover, unlike in UCT, the rate of the simple regret reduction is then exponential in the number of iterations.

### 3.3 Two-phase sampling and BRUE

While both the simple regret convergence rate, as well as the length of the transition period of **CraftyUniform**, are more attractive than those of UCT, this in itself is not much of a help: **CraftyUniform** requires explicit reasoning about  $K^{B^H}$  arms, and thus it cannot be efficiently implemented. However, it does show the promise of separation of concerns in online planning. We now introduce an **MCTS** family of algorithms, referred to as **MCTS2e**, that allows utilizing this promise to a large extent.

The instances of the **MCTS2e** family vary along four parameters: *switching point function*  $\sigma : \mathbb{N} \rightarrow \{1, \dots, H\}$ , *exploration policy*, *estimation policy*, and *update policy*. With respect to these four parameters, the **MCTS** components in **MCTS2e** are as follows.

- Similarly to UCT, each node/action pair  $(s, a)$  is associated with variables  $n(s, a)$  and  $\hat{Q}(s, a)$ . However, while counters  $n(s, a)$  are initialized to 0, value accumulators  $\hat{Q}(s, a)$  are schematically initialized to  $-\infty$ .

- **sample:** Each iteration of BRUE corresponds to a single state-space sample of the MDP, and these samples  $\rho = \langle s_0, a_1, s_1, \dots, a_k, s_k \rangle$  are all issued from the root node  $s_0$ . The sample ends either when a sink state is reached, that is,  $A(s_k) = \emptyset$ , or when  $k = H$ . The generation of  $\rho$  is done in two phases: At iteration  $n$ , the actions at states  $s_0, \dots, s_{\sigma(n)-1}$  are selected according to the exploration policy of the algorithm, while the actions at states  $s_{\sigma(n)}, \dots, s_{k-1}$  are selected according to its estimation policy.
- **expand-tree:**  $\mathcal{T}$  is expanded with the suffix of state sequence  $s_1, \dots, s_{\sigma(n)-1}$  that is new to  $\mathcal{T}$ .
- **update-statistics:** For each state  $s_i \in \{s_0, \dots, s_{\sigma(n)-1}\}$ , the update policy of the algorithm prescribes whether it should be updated. If  $s_i$  should be updated, then the counter  $n(s_i, a_{i+1})$  is incremented and the estimated  $Q$ -value is updated according to Eq. 2 (p. 4).
- **recommend-action:** The recommended action is chosen uniformly at random among the actions  $a$  maximizing  $\hat{Q}(s_0, a)$ .

In what follows, for  $n > 0$ , the  $n$ -th iteration of BRUE will be called  $\mathcal{H}$ -iteration if  $\sigma(n) = \mathcal{H}$ . At a high level, the two phases of sample generation respectively target the two exploratory objectives of online MDP planning: While the sample prefixes aim at exploring the options, the sample suffixes aim at improving the value estimates for the current candidates for  $\pi^*$ . In particular, this separation allows us to introduce a specific MCTS2e instance, BRUE,<sup>1</sup> that is tailored to simple regret minimization. The BRUE setting of MCTS2e is described below, and Figure 3 illustrates its dynamics.

- The switching point function  $\sigma : \mathbb{N} \rightarrow \{1, \dots, H\}$  is

$$\sigma(n) = H - ((n - 1) \bmod H), \quad (9)$$

that is, the depth of exploration is chosen by a round-robin on  $\{1, \dots, H\}$ , in reverse order.

- At state  $s$ , the exploration policy samples an action uniformly at random, while the estimation policy samples an action uniformly at random, but only among the actions  $a \in A(s)$  that *maximize*  $\hat{Q}(s, a)$ .
- For a sample  $\rho$  issued at iteration  $n$ , only the state/action pair  $(s_{\sigma(n)-1}, a_{\sigma(n)})$  immediately preceding the switching state  $s_{\sigma(n)}$  along  $\rho$  is updated. That is, the information obtained by the second phase of  $\rho$  is used only for improving the estimate at state  $s_{\sigma(n)-1}$ , and is *not* pushed further up the sample. While that may appear wasteful and even counterintuitive, this locality of update is required to satisfy the formal guarantees of BRUE discussed below.

Before we proceed with the formal analysis of BRUE, a few comments on it, as well as on the MCTS2e sampling scheme in general, are in place. First, the template of MCTS2e is

---

1. Short for **B**est **R**ecommendation with **U**niform **E**xploration; the name is carried on from our first presentation of the algorithm in (Feldman & Domshlak, 2012), where “estimation” was referred to as “recommendation.”

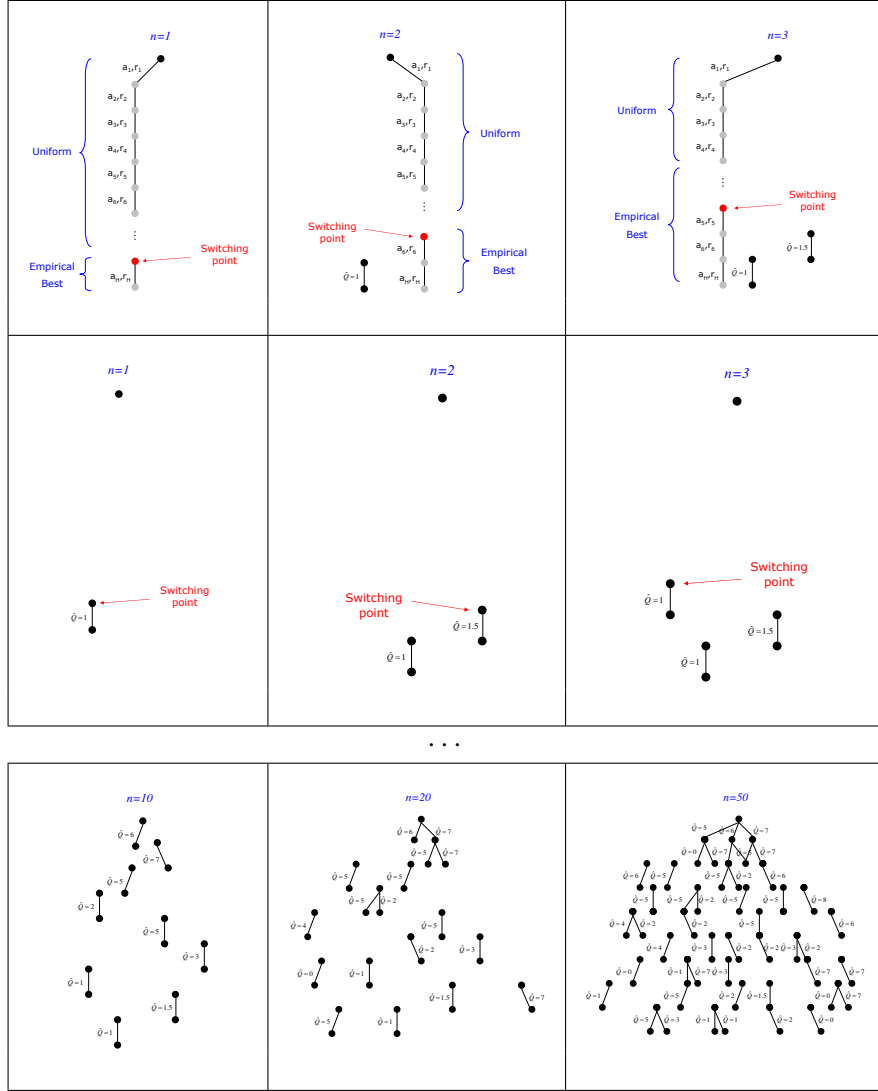


Figure 3: Illustration of the BRUE dynamics

rather general, and some of its parametrizations will not even guarantee convergence to the optimal action. This, for instance, will be the case with a (seemingly minor) modification of BRUE to purely uniform estimation policy. In short, MCTS2e should be parametrized with care. Second, while in what follows we focus on BRUE, other instances of MCTS2e may appear to be empirically effective as well with respect to the reduction of simple regret over time. Some of them, similarly to BRUE, may also guarantee exponential-rate reduction of simple regret over time. Hence, we clearly cannot, and do not, claim any uniqueness of BRUE in that respect. Finally, some other families of MCTS algorithms, more sophisticated than MCTS2e, can give rise to even more (formally and/or empirically) efficient optimizers of simple regret. The  $\text{BRUE}(\alpha)$  set of algorithms that we discuss later on is one such example.

#### 4. Upper Bounds on Simple Regret Reduction Rate with BRUE

For the sake of simplicity, in our formal analysis of BRUE we assume uniqueness of the optimal policy  $\pi^*$ ; that is, at each state  $s$  and each number  $h$  of steps-to-go, there is a single optimal action, and it is  $\pi^*(s, h)$ . Let  $\mathcal{T}_n$  be the graph obtained by BRUE after  $n$  iterations, and let  $\hat{Q}_h(s, a)$  denote the accumulated value  $\hat{Q}(s, a)$  for  $s$  at depth  $H - h$ . For all state/steps-to-go pairs  $(s, h) \in \mathcal{T}_n$ ,  $\pi_n^B(s, h)$  is a randomized strategy, uniformly choosing among actions  $a$  maximizing  $\hat{Q}_h(s, a)$ . We also use some additional auxiliary notation.

$K = \max_{s \in S} |A(s)|$ , i.e., the maximal number of actions per state.

$p = \min_{s, a, s': Tr(s, a, s') > 0} Tr(s, a, s')$ , i.e., the likelihood of the least likely (but still possible) outcome of an action in our problem.

$d = \min_{s, a} \Delta_1[s, a]$ , i.e., the smallest difference between the value of the optimal and a second-best action at a state with just one step-to-go.

Our key result on the BRUE algorithm is Theorem 1 below. The proof of Theorem 1, as well as of several required auxiliary claims, is given in Appendix A. Here we outline only the key issues addressed by the proof, and provide a high-level flow of the proof in terms of a few central auxiliary claims.

**Theorem 1** *Let BRUE be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . There exist pairs of parameters  $c, c' > 0$ , dependent only on  $\{p, d, K, H\}$ , such that, after  $n > H$  iterations of BRUE, we have simple regret bounded as*

$$\mathbb{E} \Delta_H[s, \pi_n^B(s_0, H)] \leq Hc \cdot e^{-c'n}, \quad (10)$$

and choice-error probability bounded as

$$\mathbb{P} \{ \pi_n^B(s_0, H) \neq \pi^*(s_0, H) \} \leq c \cdot e^{-c'n}. \quad (11)$$

In particular, these bounds hold for

$$c = \frac{4K^{3H^2-2H}(H!)^3 \prod_{h=1}^{H-1} (h!)^4 2^{4^{H-1}} 16^{(H-1)^2}}{d^{2H^2-4H+2} p^{3H^2-3H}}, \quad (12)$$

and

$$c' = \frac{3d^{2H-2} p^{2H-1}}{2H 16^{H-1} (H!)^2 K^{2H}}. \quad (13)$$

Before we proceed any further, some discussion of the statements in Theorem 1 are in place. First, the parameters  $c$  and  $c'$  in the bounds established by Theorem 1 are problem-dependent: in addition to the dependance on the horizon  $H$  and the choice branching factor  $K$  (which is unavoidable), the parameters  $c$  and  $c'$  also depend on the distribution parameters  $p$  and  $d$ . While it is possible that this dependence can be partly alleviated, Bubeck et al. (2011) showed that distribution-free exponential bounds on the simple regret reduction rate cannot be achieved even in MABs, that is, even in single-step-to-go MDPs (see Remark 2 of Bubeck et al. (2011), which is based on a lower bound on the cumulative

regret established by Auer, Cesa-Bianchi, Freund, & Schapire, 2002b). Second, the specific parameters  $c$  and  $c'$  provided by Eqs. 12 and 13 are worst-case for MDPs with parameters  $d$ ,  $p$ , and  $K$ , and the bound in Eq. 10 becomes effective after

$$n > \frac{\ln(c)}{c'} = O \left[ \left( \frac{KH}{pd} \right)^{\varepsilon H^2} \right]$$

iterations, for some small constant  $\varepsilon > 1$ . While there is still some gap with this transition period length and the transition period length of the theoretical **CraftyUniform** algorithm (see Eq. 8), this gap is not that large.<sup>2</sup>

The proof of Lemma 2 below constitutes the crux of the proof of Theorem 1. Once we have proven this lemma, the proof of Theorem 1 stems from it in a more-or-less direct manner.

**Lemma 2** *Let BRUE be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . For each  $h \in \llbracket H \rrbracket$ , there exist parameters  $c_h, c'_h > 0$ , dependent only on  $\{p, d, K, H\}$ , such that, for each state  $s$  reachable from  $s_0$  in  $H - h$  steps and any  $t > 0$ , it holds that*

$$\begin{aligned} \mathbb{P} \left\{ \widehat{Q}_h(s, a) - Q_h(s, a) \geq \frac{d}{2} \mid n_h(s, a) = t \right\} &\leq c_h e^{-c'_h t}, \\ \mathbb{P} \left\{ \widehat{Q}_h(s, a) - Q_h(s, a) \leq -\frac{d}{2} \mid n_h(s, a) = t \right\} &\leq c_h e^{-c'_h t}. \end{aligned} \quad (14)$$

In particular, these bounds hold for

$$c_h = \frac{K^{2Hh+h^2-2H-1} (h!)^3 \prod_{i=1}^{h-1} (i!)^4 24^{h-1} 16^{(h-1)^2}}{d^{2(h-1)^2} \cdot p^{2Hh+h^2-2H-h}}, \quad (15)$$

and

$$c'_h = \frac{3d^{2(h-1)} p^{H+h-1}}{16^{h-1} (h!)^2 K^{H+h-1}}. \quad (16)$$

The proof for Lemma 2 is by induction on  $h$ . Starting with the induction basis for  $h = 1$ , it is easy to verify that, by the Chernoff-Hoeffding inequality,

$$\mathbb{P} \left\{ \left| \widehat{Q}_1(s, a) - Q_1(s, a) \right| \geq \frac{d}{2} \mid n(s, a) = t \right\} \leq 2e^{-\frac{d^2}{2}t}, \quad (17)$$

that is, the assertion is satisfied with  $c_1 = 1$  and  $c'_1 = \frac{d^2}{2}$ . Now, assuming the claim holds for  $h \geq 1$ , below we outline the proof for  $h + 1$ , relegating the actual proof in full detail to Appendix A.

In the proof for  $h > 1$ , it is crucial to note the invalidity of applying the Chernoff-Hoeffding bound directly, as was done in Eq. 17. There are two reasons for this.

---

2. Some of this gap can probably be eliminated by more accurate bounding in the numerous bounding steps towards the proof of Theorem 1. However, all such improvements we tried made the already lengthy proof of Theorem 1 even more involved.

- (F1) For  $h = 1$ ,  $\widehat{Q}$  is an *unbiased* estimator of  $Q$ , that is,  $\mathbb{E}\widehat{Q} = Q$ . In contrast, the estimates inside the tree (at nodes with  $h > 1$ ) are biased. This bias stems from  $\widehat{Q}$  possibly being based on numerous sub-optimal choices in the sub-tree rooted in  $(s, h)$ .
- (F2) For  $h = 1$ , the summands accumulated by  $\widehat{Q}$  are independent. This is not so for  $h > 1$ , where the accumulated reward depends on the selection of actions in subsequent nodes, which in turn depends on previous rewards.

However, we show that these deficiencies of  $h > 1$  can still be overcome through a novel modification of the seminal Hoeffding-Azuma inequality.

**Lemma 3 (Modified Hoeffding-Azuma inequality)** *Let  $\{X_i\}_{i=1}^\infty$  be a sequence of random variables with support  $[0, h]$  and  $\mu_i \triangleq \mathbb{E}X_i$ . If  $\lim_{i \rightarrow \infty} \mu_i = \mu$ , and*

$$\mathbb{P}\{\mathbb{E}[X_i \mid X_1, \dots, X_{i-1}] \neq \mu\} \leq c_p e^{-c_e i}, \quad (18)$$

for some  $0 < c_p$  and  $0 < c_e \leq 1$ , then, for all  $0 < \delta \leq \frac{h}{2}$ , it holds that

$$\mathbb{P}\left\{\sum_{i=1}^t X_i \geq \mu t + t\delta\right\} \leq \left[1 + c_p \frac{2h^2}{\delta^2 c_e^2}\right] \cdot e^{-\frac{3\delta^2 c_e}{2h^2} t}, \quad (19)$$

$$\mathbb{P}\left\{\sum_{i=1}^t X_i \leq \mu t - t\delta\right\} \leq \left[1 + c_p \frac{2h^2}{\delta^2 c_e^2}\right] \cdot e^{-\frac{3\delta^2 c_e}{2h^2} t}. \quad (20)$$

Together with Lemma 4 below, the inequalities provided by Lemma 3 allow us to prove the induction hypothesis in the proof of the central Lemma 2. Note that the specific bound in Lemma 3 is selected so to maximize the exponent coefficient. For any  $0 \leq \beta \leq 1$ , the probabilities of interest in Eqs. 19-20 can also be bounded by

$$\left[1 + \frac{c_p}{c_e(1-\beta)} e^{-\frac{c_e(1-\beta)}{2h^2}}\right] e^{-\frac{3\delta^2 c_e \beta}{2h^2} t};$$

for further details, we refer the reader to Discussion 14 in Appendix A.

**Definition 1** *Let  $\langle S, A, Tr, R \rangle$  be an MDP with rewards in  $[0, 1]$ , planned for initial state  $s_0 \in S$  and finite horizon  $H$ . Let  $s$  be a state reachable from  $s_0$  with  $h$  steps still to go, let  $a$  be an action applicable in  $s$ , and let  $\pi_t^B$  be a policy induced by running BRUE on  $s_0$  until exactly  $t > 0$  samples have finished their exploration phase with applying action  $a$  at  $s$  with  $h - 1$  steps still to go. Given that,*

- $X_{t,h}(s, a)$  is a random variable, corresponding to the reward obtained by taking  $a$  at  $s$ , and then following  $\pi_t^B$  for the remaining  $h - 1$  steps.
- $E_{t,h}(s, a)$  is the event in which  $X_{t,h}(s, a)$  is sampled along the optimal actions at each of the  $h - 1$  choice points delegated to  $\pi_t^B$ .
- $\delta_{t,h}(s, a) = Q_h(s, a) - \mathbb{E}[X_{t,h}(s, a)]$ .



**Lemma 4** *Let  $\langle S, A, Tr, R \rangle$  be an MDP with rewards in  $[0, 1]$ , planned for initial state  $s_0 \in S$  and finite horizon  $H$ . Let  $s$  be a state reachable from  $s_0$  with  $h + 1$  steps still to go, and  $a$  be an action applicable in  $s$ . Considering  $E_{t,h+1}(s, a)$  and  $\delta_{t,h+1}(s, a)$  as in Definition 1, for any  $t > 0$ , if Lemma 2 holds for horizon  $h$ , then*

$$\mathbb{P}\{\neg E_{t,h+1}(s, a)\} \leq 2Kh(2 + c_h)e^{-\frac{pc'_h}{6K}t}, \quad (21)$$

$$\delta_{t,h+1}(s, a) \leq 2Kh^2(2 + c_h)e^{-\frac{pc'_h}{6K}t}. \quad (22)$$

Together with a modified version of the Hoeffding-Azuma bound in Lemma 3, the bounds established in Lemma 4 allow us to derive concentration bounds for  $\widehat{Q}_{h+1}$  around  $Q_{h+1}$  as in Lemma 5 below, which serves the key building block for proving the induction hypothesis in the proof of Lemma 2.

**Lemma 5** *Let BRUE be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . For each state  $s$  reachable  $s_0$  with  $h + 1$  steps still to go, each action  $a$  applicable, and any  $t > 0$ , it holds that*

$$\mathbb{P}\left\{\left|\widehat{Q}_{h+1}(s, a) - Q_{h+1}(s, a)\right| \geq \frac{d}{2} \mid n_{h+1}(s, a) = t\right\} \leq \left(3456 \cdot \frac{K^3(h+1)^3 c_h}{d^2 p^2 c_h^2}\right) e^{-\frac{d^2 p c'_h}{16(h+1)^2 K}}. \quad (23)$$

## 5. Learning With Forgetting and BRUE( $\alpha$ )

When we consider the evolution of action value estimates in BRUE over time (as well as in all other Monte-Carlo algorithms for online MDP planning), we can see that, in internal nodes these estimates are based on biased samples that stem from the selection of non-optimal actions at descendant nodes. This bias tends to shrink as more samples are accumulated down the tree. Consequently, the estimates become more accurate, the probability of selecting an optimal action increases accordingly, and the bias of ancestor nodes shrinks in turn. An interesting question in this context is: shouldn't we weigh differently samples obtained at different stages of the sampling process? Intuition tells us that biased samples still provide us with valuable information, especially when they are all we have, but the value of this information decreases as we obtain more and more accurate samples. Hence, in principle, putting more weight on samples with smaller bias could increase the accuracy of our estimates. The key question, of course, is which of all possible weighting schemes are both reasonable to employ and preserve the exponential-rate reduction of expected simple regret.

Here we describe BRUE( $\alpha$ ), an algorithm that generalizes BRUE  $\equiv$  BRUE(1) by basing the estimates only on the  $\alpha$  fraction of most recent samples. We discuss the value of this addition both from the perspective of the formal guarantees, as well as from the perspective of empirical prospects. BRUE( $\alpha$ ) differs from BRUE in two points:

- In addition to the variables  $n(s, a)$  and  $\widehat{Q}(s, a)$ , each node/action pair  $(s, a)$  in BRUE( $\alpha$ ) is associated with a *list*  $\mathcal{L}(s, a)$  of rewards, collected at each of the  $n(s, a)$  samples that are responsible for the current estimate  $\widehat{Q}(s, a)$ .

- When a sample  $\rho = \langle s_0, a_1, s_1, \dots, a_k, s_k \rangle$  is issued at iteration  $n$ , and **update-statistics** updates the variables at  $x = (s_{\sigma(n)-1}, a_{\sigma(n)})$ , that update is done not according to Eq. 2 as in BRUE, but according to:

$$\begin{aligned}
n(x) &\leftarrow n(x) + 1, \\
\mathcal{L}(x)[n(x)] &\leftarrow \sum_{i=\sigma(n)-1}^{k-1} R(s_i, a_{i+1}, s_{i+1}), \\
\widehat{Q}(x) &\leftarrow \frac{1}{\lceil \alpha \cdot n(x) \rceil} \sum_{i=n(x)-\lceil \alpha \cdot n(x) \rceil}^{n(x)} \mathcal{L}(x)[i].
\end{aligned} \tag{24}$$

**Theorem 6** *Let  $\text{BRUE}(\alpha)$  be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . There exist pairs of parameters  $c, c' > 0$ , dependent only on  $\{\alpha, p, d, K, H\}$ , such that, after  $n > H$  iterations of BRUE, we have simple regret bounded as*

$$\mathbb{E} \Delta_H[s, \pi_n^B(s_0, H)] \leq Hc \cdot e^{-c'n}, \tag{25}$$

*and choice-error probability bounded as*

$$\mathbb{P} \{ \pi_n^B(s_0, H) \neq \pi^*(s_0, H) \} \leq c \cdot e^{-c'n}. \tag{26}$$

The proof for Theorem 6 follows from Lemma 7 below similarly to the way Theorem 1 follows from Lemma 2. Note that in Theorem 6 we do not provide explicit expressions for the constants  $c$  and  $c'$  as we did in Theorem 1 (for  $\alpha = 1$ ). This is because the expressions that can be extracted from the recursive formulas in this case do not bring much insight. However, we discuss the potential benefits of choosing  $\alpha < 1$  in the context of our proof of Theorem 6.

**Lemma 7** *Let  $\text{BRUE}(\alpha)$  be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . For each  $h \in \llbracket H \rrbracket$ , there exist parameters  $c_h, c'_h > 0$ , dependent only on  $\{\alpha, p, d, K, H\}$ , such that, for each state  $s$  reachable from  $s_0$  in  $H - h$  steps and any  $t > 0$ , it holds that*

$$\begin{aligned}
\mathbb{P} \left\{ \widehat{Q}_h(s, a) - Q_h(s, a) \geq \frac{d}{2} \mid n_h(s, a) = t \right\} &\leq c_h e^{-c'_h t}, \\
\mathbb{P} \left\{ \widehat{Q}_h(s, a) - Q_h(s, a) \leq -\frac{d}{2} \mid n_h(s, a) = t \right\} &\leq c_h e^{-c'_h t}.
\end{aligned} \tag{27}$$

The proof for Lemma 7 is by induction, following the same line of the proof for Lemma 2. In fact, it deviates from the latter only in the application of the modified Hoeffding-Azuma inequality, which has to be further modified to capture the partial sums as in  $\text{BRUE}(\alpha)$ .

**Lemma 8 (Modified Hoeffding-Azuma inequality for partial sums)** *Let  $\{X_i\}_{i=1}^\infty$  be a sequence of random variables with support  $[0, h]$  and  $\mu_i \triangleq \mathbb{E}X_i$ . If  $\lim_{i \rightarrow \infty} \mu_i = \mu$ , and*

$$\mathbb{P} \{ \mathbb{E}[X_i \mid X_1, \dots, X_{i-1}] \neq \mu \} \leq c_p e^{-c_e i}, \tag{28}$$

for some  $0 < c_p$  and  $0 < c_e \leq 1$ , then, for all  $0 < \delta \leq \frac{h}{2}$ , it holds that

$$\mathbb{P} \left\{ \sum_{i=t-\lceil \alpha t \rceil}^t X_i \geq \mu t + t\delta \right\} \leq \left[ 1 + \frac{c_p}{c_e(1-\alpha)} e^{-c_e(1-\alpha)^2 t} \right] e^{-\frac{3\delta^2 c_e}{2h^2} \alpha t}, \quad (29)$$

$$\mathbb{P} \left\{ \sum_{i=t-\lceil \alpha t \rceil}^t X_i \leq \mu t - t\delta \right\} \leq \left[ 1 + \frac{c_p}{c_e(1-\alpha)} e^{-c_e(1-\alpha)^2 t} \right] e^{-\frac{3\delta^2 c_e}{2h^2} \alpha t}. \quad (30)$$

Considering the benefits of “sample forgetting” as in  $\text{BRUE}(\alpha)$ , let us compare the bound in Lemma 8 to the bound

$$e^{-\frac{3\delta^2 c_e \beta}{2h^2} t} \left[ 1 + \frac{c_p}{c_e(1-\beta)} e^{-\frac{c_e(1-\beta)}{2h^2} t} \right],$$

provided by Lemma 3 for  $\text{BRUE}$ , that is, when *all* accumulated samples are averaged. While both bounds are very similar, the exponent of the second exponential term is multiplied for  $\text{BRUE}(\alpha < 1)$  by  $(1-\alpha)t$ . This poses a tradeoff: Decreasing  $\alpha$  reduces the sampling bias, and thus decreases the term  $\frac{c_p}{c_e}$ , but increases the other exponential term with no leading constant. Obviously, since there is no bias at leaf nodes, it makes no sense to set  $\alpha < 1$  there. However, as we go further up the tree, the bias tends to grow ( $\frac{c_p}{c_e} \gg 1$ ), but we also expect to have more samples ( $t$  is larger). Thus, from the perspective of formal guarantees, it seems appealing to choose smaller values of  $\alpha$ . Nevertheless, we do not try to optimize here the value of  $\alpha$ : First, optimizing bounds doesn’t necessarily lead to optimized empirical accuracy. Second, the underlying optimization would have to be specific to each horizon  $h$  and each sample size  $t$  (which is obviously out of the question), and thus anyway we would have to consider only some rough approximations to this optimization problem. Finally, biased samples in practice might be more valuable than what the theory suggests, as long as all actions at the same state/steps-to-go decision point experience a similar bias.

## 6. Experimental Evaluation

We have evaluated  $\text{BRUE}$  empirically on the MDP sailing domain (Péret & Garcia, 2004) that was used in previous works for evaluating MC planning algorithms (Péret & Garcia, 2004; Kocsis & Szepesvári, 2006; Tolpin & Shimony, 2012), as well as on random game trees used in the original empirical evaluation of UCT (Kocsis & Szepesvári, 2006).

In the sailing domain, a sailboat navigates to a destination on an 8-connected grid representing a marine environment, under fluctuating wind conditions. The goal is to reach the destination as quickly as possible, by choosing at each grid location a neighbor location to move to. The duration of each such move depends on the direction of the move (*ceteris paribus*, diagonal moves take  $\sqrt{2}$  more time than straight moves), the direction of the wind relative to the sailing direction (the sailboat cannot sail against the wind and moves fastest with a tail wind), and the tack. The direction of the wind changes over time, but its strength is assumed to be fixed. This sailing problem can be formulated as a goal-driven MDP over finite state space and a finite set of actions, with each state capturing the position of the sailboat, wind direction, and tack.

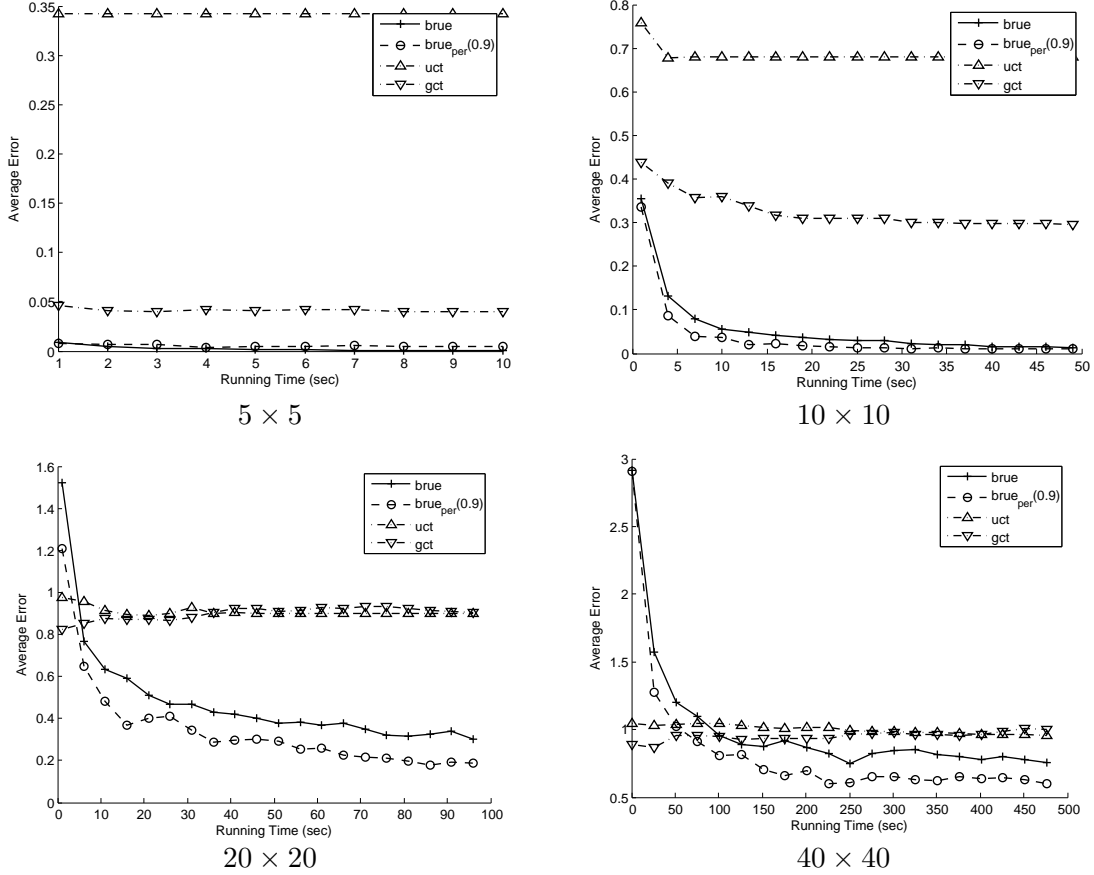


Figure 4: Empirical performance of BRUE, BRUE(0.9), UCT, and  $\epsilon$ -greedy + UCT (denoted as GCT, for short) in terms of the average error on sailing domain problems on  $n \times n$  grids with  $n \in \{5, 10, 20, 40\}$ .

In a goal-driven MDP, the lengths of the paths to a terminal state are not necessarily bounded, and thus it is not entirely clear to what depth BRUE shall construct its tree. In the sailing domain, we chose  $H$  to be  $4 \times n$ , where  $n$  is the grid-size of the problem instance, as it is unlikely that the optimal path between any two locations on the grid will be larger than a complete encircling of the considered area. We note, however, that the recommendation-oriented samples  $\bar{\rho}$  always end at a terminal state, similar to the rollouts issued by UCT and  $\epsilon$ -greedy + UCT.

Snapshots of the results for different grid sizes are shown in Figure 4. We compared BRUE with two MCTS-based algorithms: the UCT algorithm, and a recent modification of UCT,  $\epsilon$ -greedy + UCT, obtained from the former by replacing the UCB1 policy *at the root node* with the  $\epsilon$ -greedy policy (Tolpin & Shimony, 2012). The motivation behind the design of  $\epsilon$ -greedy + UCT was to improve the empirical simple regret of UCT, and the results for  $\epsilon$ -greedy + UCT reported by (Tolpin & Shimony, 2012) (and confirmed by our experiments

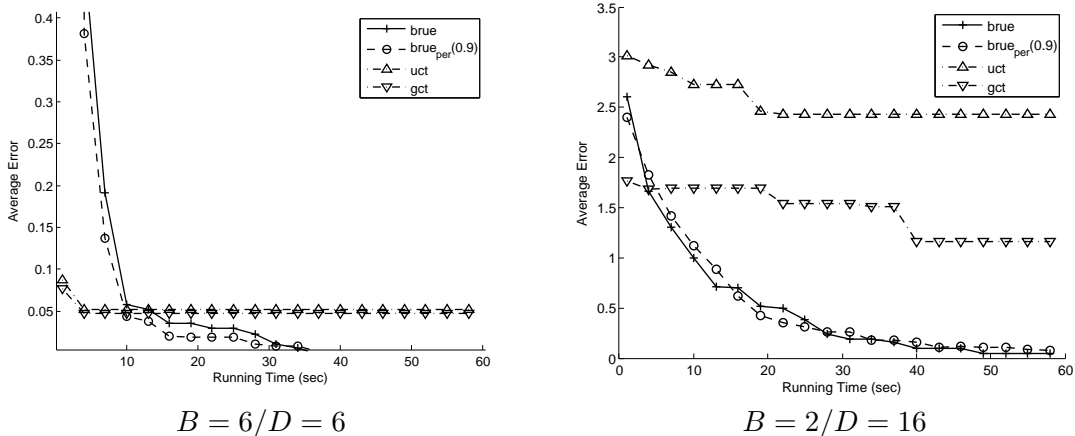


Figure 5: Empirical performance of BRUE, UCT, and  $\epsilon$ -greedy + UCT (denoted as GCT) in terms of the average error on the random game trees with branching factor  $B$  and tree depth  $D$ .

here) are very impressive. We also show the results for  $\text{BRUE}_{\text{per}}(0.9)$ , a slight modification of  $\text{BRUE}(0.9)$  with a more permissive update scheme: Instead of updating only the state-action node at the level of the switching point, we also update any ancestor for which either not all applicable actions have been sampled or the chosen action was identical to the best empirical one.

All four algorithms were implemented within a single software infrastructure. As suggested by more recent works on UCT, the exploration coefficient for UCT and  $\epsilon$ -greedy + UCT (parameter  $c$  in Eq. 1) was set to the empirical best value of an action at the decision point (Keller & Eyerich, 2012b). (This setting of the exploration coefficient resulted in better performance of both UCT and  $\epsilon$ -greedy + UCT than with the settings reported on the sailing domain in the respective original publications.) The  $\epsilon$  parameter in  $\epsilon$ -greedy + UCT was set to 0.5 as in the experiments of Tolpin & Shimony, 2012. Each algorithm was run on 1000 randomly chosen initial states  $s_0$ , and the performance of the algorithm was assessed in terms of the average error  $Q(s_0, a) - V(s_0)$ , that is, the difference between the true values of the action  $a$  chosen by the algorithm and that of the optimal action  $\pi^*(s_0)$ . Consistently with the results reported by Tolpin and Shimony (2012), on the smaller tasks  $\epsilon$ -greedy + UCT outperformed UCT by a very large margin, with the latter exhibiting very little improvement over time even on the smallest,  $5 \times 5$ , grids. The difference between  $\epsilon$ -greedy + UCT and UCT on the larger tasks was less notable. In turn, BRUE substantially outperformed  $\epsilon$ -greedy + UCT, with the improvement being consistent except for relatively short planning deadlines, and  $\text{BRUE}_{\text{per}}(0.9)$  performed even better than BRUE.

The above allows us to conclude that BRUE is not only attractive in terms of the formal performance guarantees, but can also be very effective in practice for online planning. Likewise, the “learning with forgetting” extension of  $\text{BRUE}(\alpha)$  also has its practical merits. Under the same parameter setting of UCT and  $\epsilon$ -greedy + UCT, we have also evaluated the

three algorithms in a domain of random game trees whose goal is a simple modeling of two-person zero-sum games such as Go, Amazons and Globber. In such games, the winner is decided by a global evaluation of the end board, with the evaluation employing this or another feature counting procedure; the rewards thus are associated only with the terminal states. The rewards are calculated by first assigning values to moves, and then summing up these values along the paths to the terminal states. Note that the move values are used for the tree construction only and are not made available to the players. The values are chosen uniformly from  $[0, 127]$  for the moves of MAX, and from  $[-127, 0]$  for the moves of MIN. The players act so to (depending on the role) maximize/minimize their individual payoff: the aim of MAX is to reach terminal  $s$  with as high  $R(s)$  as possible, and the objective of MIN is similar, *mutatis mutandis*. This simple game tree model is similar in spirit to many other game tree models used in previous work (Kocsis & Szepesvári, 2006; Smith & Nau, 1994), except that the success/failure of the players is measured not on a ternary scale of win/lose/draw, but via the actual payoffs they receive. We ran some experiments with two different settings of the branching factor ( $B$ ) and tree depths ( $D$ ). As in the sailing domain, we compared the convergence rate obtained by BRUE, UCT and  $\epsilon$ -greedy + UCT. Figure 5 plots the average error rate for two configurations,  $B = 6, D = 6$  and  $B = 2, D = 16$ , with the average in each setting obtained over 500 trees. The results here appear encouraging as well, with BRUE overtaking the other two algorithms more quickly on the deeper trees.

## 7. SUMMARY

We have introduced BRUE, a simple Monte-Carlo algorithm for online planning in MDPs that guarantees exponential-rate reduction of the performance measures of interest, namely the simple regret and the probability of erroneous action choice. This improves over previous algorithms such as UCT, which guarantee only polynomial-rate reduction of these measures. The algorithm has been formalized for finite horizon MDPs, and it was analyzed as such. However, our empirical evaluation shows that it also performs well on goal-driven MDPs and two-person games.

A few questions remain for future work. In the setting of  $\gamma$ -discounted MDPs with infinite horizons, a straightforward way to employ BRUE is to fix a horizon  $H$ , use the algorithm as is, and derive guarantees on the aforementioned measures of interest by simply accounting for the additive gap of  $\gamma^H R_{\max}/(1 - \gamma)$  between the state/action values under horizon  $H$  and those under an infinite horizon. However, this is not necessarily the best way to plan online for infinite-horizon MDPs, and thus this setting requires further inspection. Second, it is not unlikely that the state-space independent factors  $c_h$ , and  $c'_h$  in the guarantees of BRUE can be improved by employing more sophisticated combinations of exploration and estimation samples. Another important point to consider is the speed of convergence to the optimal action, as opposed to the speed of convergence to “good” actions. BRUE is geared towards identifying the optimal action, although in many large MDPs, “good” is often the best one can hope for. To identify the optimal solution, BRUE devotes samples equally to all depths. However, focusing on nodes closer to the root node may improve the quality of the recommendation if the planning time is severely limited. Finally, the core tree sampling scheme employed by BRUE differs from the more standard scheme employed in previous work. While this difference plays a critical role in establishing

the formal guarantees of BRUE, it is still unclear whether that difference is *necessary* for establishing exponential-over-time reduction of the performance measures.

## Acknowledgements

This work is partially supported by and carried out at the Technion-Microsoft Electronic Commerce Research Center, as well as partially supported by the Air Force Office of Scientific Research, USAF, under grant number FA8655-12-1-2096.

## References

- Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3), 235–256.
- Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32(1), 48–77.
- Balla, R., & Fern, A. (2009). UCT for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 40–45.
- Bjarnason, R., Fern, A., & Tadepalli, P. (2009). Lower bounding Klondike Solitaire with Monte-Carlo planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Bonet, B., & Geffner, H. (2012). Action selection for MDPs: Anytime  $ao^*$  vs. uct. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*.
- Bubeck, S., & Munos, R. (2010). Open loop optimistic planning. In *Proceedings of the 23rd Annual Conference on Learning Theory (COLT)*, pp. 477–489.
- Bubeck, S., Munos, R., & Stoltz, G. (2011). Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19), 1832–1852.
- Busoniu, L., & Munos, R. (2012). Optimistic planning for Markov decision processes. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, No. 22 in Journal of Machine Learning Research - Proceedings Track, pp. 182–189.
- Cazenave, T. (2009). Nested Monte-Carlo search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 456–461, Pasadena, California.
- Coquelin, P.-A., & Munos, R. (2007). Bandit algorithms for tree search. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 67–74, Vancouver, BC, Canada.
- Eyerich, P., Keller, T., & Helmert, M. (2010). High-quality policies for the Canadian Traveler’s problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*.
- Feldman, Z., & Domshlak, C. (2012). Online planning in mdps: Rationality and optimization. *CoRR*, arXiv:1206.3382v1 [cs.AI].

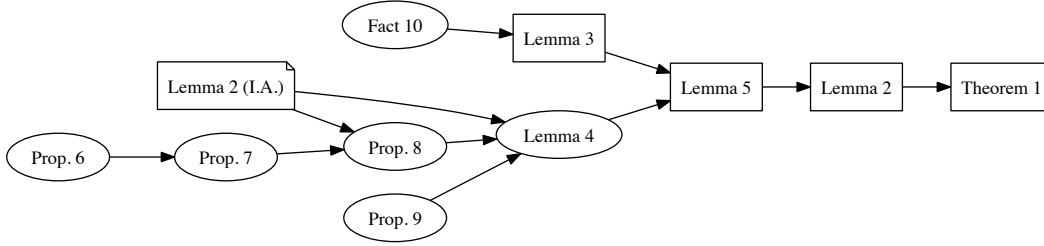
- Gelly, S., & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11), 1856–1875.
- Hay, N., Shimony, S. E., Tolpin, D., & Russell, S. (2012). Selecting computations: Theory and applications. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI)*.
- Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1324–1231, Stockholm, Sweden.
- Keller, T., & Eyerich, P. (2012a). Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.
- Keller, T., & Eyerich, P. (2012b). PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 119–127.
- Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pp. 282–293, Berlin, Germany.
- Kolobov, A., Mausam, & Weld, D. (2012). LRTDP vs. UCT for online probabilistic planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*.
- Péret, L., & Garcia, F. (2004). On-line search for solving Markov decision processes via heuristic sampling. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pp. 530–534, Valencia, Spain.
- Puterman, M. (1994). *Markov Decision Processes*. Wiley, New-York.
- Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5), 527535.
- Rosin, C. D. (2011). Nested rollout policy adaptation for Monte Carlo tree search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 649–654, Barcelona, Catalonia, Spain.
- Smith, S. J., & Nau, D. S. (1994). An analysis of forward pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1386–1391.
- Sturtevant, N. (2008). An analysis of UCT in multi-player games. In *Proceedings of the 6th International Conference on Computers and Games (CCG)*, p. 3749.
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Tolpin, D., & Shimony, S. E. (2012). MCTS based on simple regret. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*.



# Appendices

## Appendix A. Proof of Theorem 1

The proof of Theorem 1 relies on the inductive assumption with respect to the correctness of Lemma 2, as well as on several auxiliary claims that we prove in what follows. The dependence diagram below depicts the overall flow of the proof, with the more central claims being depicted with rectangular nodes.



**Proposition 9 (Concentration inequality for negative-binomial distributions)** *Let  $NB(t, p)$  be a random variable with negative-binomial distribution.*

$$\mathbb{P} \left\{ NB(t, p) \leq \frac{3t}{4p} \right\} \leq e^{-\frac{tp}{6}} \quad (31)$$

**Proof:** It is well known that the event in which the number of Bernoulli trials required to obtain the  $t$ -th success is smaller than some positive integer  $b$  is equivalent to the event that the number of successes in  $b$  Bernoulli trials is at least  $t$ . Therefore, for any  $0 < \delta < 1$ ,

$$\begin{aligned} \mathbb{P} \left\{ NB(t, p) \leq \delta \frac{t}{p} \right\} &= \mathbb{P} \left\{ \text{Bin} \left( \delta \frac{t}{p}, p \right) \geq t \right\} \\ &= \mathbb{P} \left\{ \text{Bin} \left( \delta \frac{t}{p}, p \right) \geq t\delta + (t - t\delta) \right\} \\ &\leq e^{-\frac{2t^2(1-\delta)^2p}{t\delta}} \\ &\text{by the Hoeffding inequality,} \end{aligned} \quad (32)$$

and choosing  $\delta = \frac{3}{4}$  yields the result. ■

**Proposition 10 (Number of Child Samples Bound)** *Let BRUE be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . Let  $(s, h)$  be a node reachable from  $(s_0, H)$ , and in turn,  $(s', h')$  be a node reachable from  $(s, h)$  via an action sequence that starts with applying action  $a$  at  $s$ . Then, for any  $a' \in A(s')$ , we have*

$$\mathbb{P} \left\{ n_{h'}(s', a') \leq \frac{p_{h'}(s', a')}{4p_h(s, a)} t \mid n_h(s, a) = t \right\} \leq 2e^{-t \frac{p_{h'}(s', a')^2}{6p_h(s, a)}}, \quad (33)$$

where  $p_h(s, a)$  is the probability that an  $(H - h)$ -iteration of BRUE will issue a sample, whose exploration phase ends with applying action  $a$  at state  $s$  with  $h$  steps still to go.

**Proof:** By the choice of the switching point function of BRUE as in Eq. 9, the number of samples of action  $a'$  in the descendant node  $(s', h')$  between two consecutive samples of action  $a$  in node  $(s, h)$  is distributed according to

$$\sum_{i=1}^{1+\gamma} \beta_i, \quad (34)$$

where  $\gamma \sim \text{Geo}(p_h(s, a))$  and  $\beta_i \sim \text{Ber}(p_{h'}(s', a'))$  are all independent random variables. Indeed, for every pair of consecutive iterations  $n < n'$  with  $\sigma(n) = \sigma(n') = H - h$ ,

- (i) there is exactly one iteration  $n < n'' < n'$  with  $\sigma(n'') = H - h'$ , and
- (ii) the number of  $(H - h)$ -iterations between two consecutive  $(H - h)$ -iterations that finish their exploration phase with applying action  $a$  at  $s$  is geometric.

Putting (i) and (ii) together, the number of  $(H - h')$ -iterations between a pair of consecutive  $(H - h)$ -iterations that finish their exploration phase with applying action  $a$  at  $s$  is also geometric. In turn, the probability that an  $(H - h')$ -iteration will finish its exploration phase with applying  $a'$  at  $s'$  is  $p_{h'}(s', a')$ , and thus the number of  $(H - h')$ -iterations that finish their exploration phase with applying  $a'$  at  $s'$  between a pair of consecutive  $(H - h)$ -iterations that finish their exploration phase with applying action  $a$  at  $s$  is distributed as in Eq. 34.

Similarly, it can be shown that the (conditioned) random variable

$$n_{h'}(s', a') \mid n_h(s, a) = t$$

is distributed according to

$$\sum_{i=1}^{t+\gamma_t} \beta_i, \quad (35)$$

where  $\gamma_t \sim \text{NB}(t, p_h(s, a))$ ,  $\beta_i \sim \text{Ber}(p_{h'}(s', a'))$ , and all  $\gamma_t$  and  $\beta_i$  are independent.

Therefore, denoting  $p_h(s, a)$  and  $p_{h'}(s', a')$  by  $p_h$  and  $p_{h'}$ , respectively, for short, we have

$$\begin{aligned} & \mathbb{P} \left\{ n_{h'}(s, a) \leq \frac{t}{4p_h} p_{h'} \mid n_{h'}(s, a) = t \right\} \\ & \stackrel{\text{Eq. 35}}{=} \mathbb{P} \left\{ \sum_{i=1}^{t+\gamma_t} \beta_i \leq \frac{t}{4p_h} p_{h'} \right\} \\ & \leq \mathbb{P} \left\{ \gamma_t \leq \frac{3t}{4p_h} \right\} + \sum_{x=\frac{3t}{4p_h}+1}^{\infty} \mathbb{P} \left\{ \sum_{i=1}^{t+x} \beta_i \leq \frac{t}{4p_h} p_{h'} \right\} \mathbb{P} \{ \gamma_t = x \} \\ & = \mathbb{P} \left\{ \gamma_t \leq \frac{3t}{4p_h} \right\} + \sum_{x=\frac{3t}{4p_h}+1}^{\infty} \mathbb{P} \left\{ \text{Bin}((t+x), p_{h'}) \leq \frac{t}{4p_h} p_{h'} \right\} \mathbb{P} \{ \gamma_t = x \} \end{aligned} \quad (36)$$

since  $\beta_i$  are all independent Bernoulli variables with common parameter  $p_{h'}$

$$= \mathbb{P} \left\{ \gamma_t \leq \frac{3t}{4p_h} \right\} + \sum_{x=\frac{3t}{4p_h}+1}^{\infty} \mathbb{P} \{ \text{Bin}((t+x), p_{h'}) \leq (t+x)p_{h'} - \delta_x \} \mathbb{P} \{ \gamma_t = x \},$$

where  $\delta_x = \frac{4xp_h - t(1-4p_h)}{4p_h} p_{h'}$ .

Given that, for all  $x \geq \frac{3t}{4p_h}$ , we have

$$\begin{aligned}
\mathbb{P}\{\text{Bin}((t+x), p_{h'}) \leq (t+x)p_{h'} - \delta_x\} &\leq e^{-\frac{2\delta_x^2}{(t+x)}} \\
&\text{by the Hoeffding inequality, applicable here since } \delta_x = \frac{4xp_h - t(1-4p_h)}{4p_h} p_{h'} \geq \frac{t(2+4p_h)}{4p_h} p_{h'} \geq 0 \\
&\leq e^{-t\left(2+\frac{1}{2p_h}\right)p_{h'}^2} \\
&\text{since } \frac{\delta_x^2}{t+x} \geq t + \frac{tp_{h'}^2}{4p_h} \\
&\leq e^{-\frac{t}{2p_h}p_{h'}^2}.
\end{aligned} \tag{37}$$

Plugging Eq. 37 into Eq. 36, we obtain

$$\begin{aligned}
&\mathbb{P}\left\{n_g(s, a) \leq \frac{t}{4p_h} p_{h'} \mid n_g(s, a) = t\right\} \\
&\leq \mathbb{P}\left\{\gamma_t \leq \frac{3t}{4p_h}\right\} + \sum_{x=\frac{3t}{4p_h}+1}^{\infty} e^{-\frac{p_{h'}^2 t}{2p_h}} \mathbb{P}\{\gamma_t = x\} \\
&\stackrel{\text{Prop. 9}}{\leq} e^{-\frac{tp_h}{6}} + \sum_{x=\frac{3t}{4p_h}+1}^{\infty} e^{-\frac{p_{h'}^2 t}{2p_h}} \mathbb{P}\{\gamma_t = x\} \\
&\leq e^{-\frac{tp_h}{6}} + e^{-\frac{tp_{h'}^2}{2p_h}} \\
&\leq 2e^{-\frac{tp_{h'}^2}{6p_h}}.
\end{aligned} \tag{38}$$

■

**Proposition 11** *Let BRUE be called on a state  $s_0$  of an MDP  $\langle S, A, Tr, R \rangle$  with rewards in  $[0, 1]$  and finite horizon  $H$ . Let  $(s, h+1)$  be a node reachable from  $(s_0, H)$ , and in turn,  $(s', h')$  be a node reachable from  $(s, h+1)$ . If Lemma 2 holds for horizon  $h$ , then, for any  $a \in A(s)$ ,  $a' \in A(s')$ , and  $t \geq 1$ ,*

$$\mathbb{P}\left\{\widehat{Q}_{h'}(s', a') - Q_{h'}(s', a') \geq \frac{d}{2} \mid n_{h+1}(s, a) = t\right\} \leq (2 + c_{h'}) e^{-tc'_{h'} \frac{p^{h+1-h'}}{6K^{h+1-h'}}}, \tag{39}$$

and

$$\mathbb{P}\left\{\widehat{Q}_{h'}(s', a') - Q_{h'}(s', a') \leq -\frac{d}{2} \mid n_{h+1}(s, a) = t\right\} \leq (2 + c_{h'}) e^{-tc'_{h'} \frac{p^{h+1-h'}}{6K^{h+1-h'}}}. \tag{40}$$

**Proof:** The proof for the two pairs of equations is identical, and thus we explicitly prove here only Eq. 39. In what follows, we use  $p_h(s, a)$  and  $p_{h'}(s', a')$  as defined in Proposition 10, and here as well denote them by  $p_h$  and  $p_{h'}$ , respectively, for short. Similarly, by  $Q_{h'}$ ,  $\widehat{Q}_{h'}$ ,  $n_{h'}$ , and  $n_{h+1}$  we refer to  $Q_{h'}$ ,  $\widehat{Q}_{h'}(s', a')$ ,  $n_{h'}(s', a')$ , and  $n_{h+1}(s, a)$ , respectively, for short.

$$\begin{aligned}
& \mathbb{P} \left\{ \widehat{Q}_{h'} - Q_{h'} \geq \frac{d}{2} \mid n_{h+1} = t \right\} \\
& \leq \mathbb{P} \left\{ n_{h'} \leq \frac{tp_{h'}}{4p_{h+1}} \mid n_{h+1} = t \right\} + \mathbb{P} \left\{ \widehat{Q}_{h'} - Q_{h'} \geq \frac{d}{2}, n_{h'} > \frac{tp_{h'}}{4p_{h+1}} \mid n_{h+1} = t \right\} \\
& \stackrel{\text{Prop. 10}}{\leq} 2e^{-\frac{p_{h'}^2 t}{6p_{h+1}}} + \sum_{\tau=\frac{tp_{h'}}{4p_{h+1}}}^{\infty} \mathbb{P} \left\{ \widehat{Q}_{h'} - Q_{h'} \geq \frac{d}{2} \mid n_{h'} = \tau \right\} \mathbb{P} \{ n_{h'} = \tau \mid n_{h+1} = t \} \\
& \stackrel{\text{I.A./Eq. 14}}{\leq} 2e^{-\frac{tp_{h'}^2}{6p_{h+1}}} + \sum_{\tau=\frac{tp_{h'}}{4p_{h+1}}}^{\infty} c_{h'} e^{-\tau c'_{h'}} \mathbb{P} \{ n_{h'} = \tau \mid n_{h+1} = t \} \\
& \leq 2e^{-\frac{tp_{h'}^2}{6p_{h+1}}} + c_{h'} e^{-c'_{h'} \frac{tp_{h'}}{4p_{h+1}}}.
\end{aligned} \tag{41}$$

Consider the fraction  $\frac{p_{h'}}{p_{h+1}}$ , and recall that  $(s', h')$  is a descendant of  $(s, h)$ . The latter implies that

$$p_{h'} \geq p_{h+1} \left( \frac{p}{K} \right)^{h+1-h'},$$

and thus  $\frac{p_{h'}}{p_{h+1}} \geq \left( \frac{p}{K} \right)^{h+1-h'}$ . Continuing now Eq. 41, by Eq. 16,

$$c'_{h'} = \frac{3d^{2(h'-1)}p^{H+h'-1}}{32^{h'-1}((h')!)^2 K^{H+h'-1}} < \left( \frac{p}{K} \right)^{H+h'-1} \leq \left( \frac{p}{K} \right)^{H-h'} \leq p_{h'},$$

and Eq. 41 under  $c'_{h'} < p_{h'}$  implies

$$\begin{aligned}
2e^{-\frac{tp_{h'}^2}{6p_{h+1}}} + c_{h'} e^{-c'_{h'} \frac{tp_{h'}}{4p_{h+1}}} & \leq (2 + c_{h'}) e^{-tc'_{h'} \frac{p_{h'}}{6p_{h+1}}} \\
& \leq (2 + c_{h'}) e^{-tc'_{h'} \frac{p^{h+1-h'}}{6K^{h+1-h'}}}.
\end{aligned} \tag{42}$$

■

**Proposition 12 (Expected accumulated rewards)** *Let  $\langle S, A, Tr, R \rangle$  be an MDP, and let  $X$  be the accumulated reward of a sample*

$$\rho = \langle s, a, s_1, a_1, s_h, a_h, s_{h+1} \rangle,$$

*started with taking action  $a \in A$  in state  $s \in S$ , and continued with additional  $h$  steps, in which actions are chosen according to some arbitrary (possibly randomized) policy  $\pi$ . Let*

- $E_{\pi, h+1}(s, a)$  denote the event in which, after  $a$ ,  $\rho$  is sampled along the optimal actions, that is, for  $i \in \llbracket h \rrbracket$ ,  $a_i = \pi_{h+1-i}^*(s_i)$ , and
- $\delta_{\pi, h+1}(s, a) = Q_{h+1}(s, a) - \mathbb{E}[X]$ .

Then,

$$\mathbb{P}\{\neg E_{\pi,h+1}(s,a)\} \leq \sum_{i=1}^h \mathbb{P}\{\pi_{h+1-i}(s_i) \neq \pi_{h+1-i}^*(s_i)\}, \quad (43)$$

$$\delta_{\pi,h+1}(s,a) = \sum_{i=1}^h \mathbb{E}[\Delta_{h+1-i}[s_i, \pi_{h+1-i}(s_i)]]. \quad (44)$$

**Proof:** The proof of Eq. 43 is straightforward by the union bound. To prove Eq. 44, we note that for any state/steps-to-go pair  $(s, h) \in S \times \llbracket H \rrbracket$ , we have

$$\mathbb{E}_{\pi,s'}[R(s, \pi_h(s), s')] = \mathbb{E}_{\pi}[Q_h(s, \pi_h(s))] - \mathbb{E}_{\pi,s'}[Q_{h-1}(s', \pi^*(s', h-1))].$$

Using that, we obtain a telescopic series that yields

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{E}_{\pi,s_1:s_h} \left[ R(s, a, s_1) + \sum_{i=1}^h R(s_i, \pi_{h+1-i}(s_i), s_{i+1}) \right] \\ &= Q_{h+1}(s, a) - \mathbb{E}_{s_1}[Q_h(s_1, \pi^*(s_1, h))] + \\ &\quad \sum_{i=1}^h (\mathbb{E}_{\pi,s_1:s_i}[Q_{h+1-i}(s_i, \pi_{h+1-i}(s_i))] - \mathbb{E}_{\pi,s_1:s_{i+1}}[Q_{h-i}(s_{i+1}, \pi^*(s_{i+1}, h-i))]) \\ &= Q_{h+1}(s, a) - \sum_{i=1}^h \mathbb{E}_{\pi,s_1:s_i}[\Delta_{h+1-i}[s_i, \pi_{h-i+1}(s_i)]]. \end{aligned}$$

■

#### Proof of Lemma 4:

By Definition 1, the event  $E_{t,h+1}(s,a)$  corresponds to a sample

$$\rho = \langle s, a, s_1, a_1, s_h, a_h, s_{h+1} \rangle,$$

obtained by taking action  $a$  at state  $s$ , reachable from  $s_0$  with  $h+1$  steps still to go, and then following the policy  $\pi_t^B$ , induced by running BRUE on  $s_0$  until exactly  $t > 0$  samples finish their exploration phase with applying action  $a$  at  $s$  with  $h$  steps still to go. From

Proposition 12, denoting  $\hat{i} \triangleq h + 1 - i$ , we have

$$\begin{aligned}
\mathbb{P}\{-E_{t,h+1}(s,a)\} &\leq \sum_{i=1}^h \mathbb{P}\{\pi_{\hat{i}}(s_i) \neq \pi_i^*(s_i) \mid n_{h+1}(s,a) = t\} \\
&\leq \sum_{i=1}^h \sum_{a' \neq \pi_i^*(s_i)} \mathbb{P}\left\{\widehat{Q}_i(s_i, a') > \widehat{Q}_i(s_i, \pi_i^*(s_i)) \mid n_{h+1}(s,a) = t\right\} \\
&\leq \sum_{i=1}^h \sum_{a' \neq \pi_i^*(s_i)} \left[ \mathbb{P}\left\{\widehat{Q}_i(s_i, a') - Q_i(s_i, a) \geq \frac{d}{2} \mid n_{h+1}(s,a) = t\right\} + \right. \\
&\quad \left. \mathbb{P}\left\{\widehat{Q}_i(s_i, \pi_i^*(s_i)) - Q_i(s_i, \pi_i^*(s_i)) \leq -\frac{d}{2} \mid n_{h+1}(s,a) = t\right\} \right] \\
&\stackrel{\text{Prop. 11}}{\leq} \sum_{i=1}^h 2K(2 + c_i) e^{-tc'_i \frac{p^i}{6K^i}} \\
&\stackrel{(*)}{\leq} 2Kh(2 + c_h) e^{-tc'_h \frac{p}{6K}}.
\end{aligned} \tag{45}$$

The last inequality  $(*)$  in Eq. 45 holds because, by assuming Lemma 2 for horizon  $h$ , for  $i \in \llbracket h \rrbracket$ , it can be straightforwardly derived from Eqs. 15 and 16 that  $c_i > c_{i-1}$  and  $c'_i < \frac{p}{K} c'_{i-1}$ .

Similarly,

$$\begin{aligned}
\delta_{t,h+1}(s,a) &\leq \sum_{i=1}^h \mathbb{E} \Delta_i[s_i, \pi_t^B(s_i)] \\
&\leq \sum_{i=1}^h \left[ i \sum_{a' \neq \pi_i^*(s_i)} \mathbb{P}\left\{\widehat{Q}_i(s_i, a') > \widehat{Q}_i(s_i, \pi_i^*(s_i)) \mid n_{h+1}(s,a) = t\right\} \right] \\
&\leq \sum_{i=1}^h \left[ h \sum_{a' \neq \pi_i^*(s_i)} \mathbb{P}\left\{\widehat{Q}_i(s_i, a') > \widehat{Q}_i(s_i, \pi_i^*(s_i)) \mid n_{h+1}(s,a) = t\right\} \right] \\
&\stackrel{\text{Prop. 11}}{\leq} \sum_{i=1}^h 2Kh(2 + c_i) e^{-tc'_i \frac{p^i}{6K^i}} \\
&\leq 2Kh^2(2 + c_h) e^{-tc'_h \frac{p}{6K}}.
\end{aligned} \tag{46}$$

■

**Fact 13** Let  $Z$  be a random variable with support  $[a, b]$  and  $\mathbb{E}[Z] = 0$ . Then, for any  $\lambda \in \mathbb{R}^+$ ,

$$\mathbb{E}[\exp(\lambda Z)] \leq \exp\left(\frac{(b-a)^2 \lambda^2}{8}\right).$$

This result is well known due to Hoeffding.

**Proof of Lemma 3 (Modified Hoeffding-Azuma inequality):**

Let  $E_t$  be the event that  $\mathbb{E}[X_t \mid X_1, \dots, X_{t-1}] = \mu$ , and let

$$Y_t \triangleq X_t - \mu \mid X_1(\omega), \dots, X_{t-1}(\omega).$$

The random variable  $Y_t$  is bounded by  $h - \mu \geq Y_t \geq -\mu$ , and furthermore, for  $\omega \in E_t$ ,  $\mathbb{E}Y_t = 0$ . Therefore, using Fact 13, for all  $\omega \in E_t$  and  $\lambda \in \mathbb{R}^+$ , it holds that

$$\mathbb{E}\left[e^{\lambda Y}\right] \leq e^{\frac{\lambda^2 h^2}{8}}. \quad (47)$$

Moreover,

$$\begin{aligned} & \mathbb{E}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right] \\ &= \mathbb{E}_{E_t}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right] \\ &= \mathbb{E}_{E_t}\left[\mathbb{E}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)} \mid X_1, \dots, X_{t-1}\right]\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right] \\ &= \mathbb{E}\left[e^{\lambda \sum_{i=1}^{t-1} (\mu - X_i)} \cdot \mathbb{E}\left[e^{\lambda (\mu - X_t)} \mid X_1, \dots, X_{t-1}\right]\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right] \\ &\leq e^{\frac{\lambda^2 h^2}{8}} \mathbb{E}\left[e^{\lambda \sum_{i=1}^{t-1} (\mu - X_i)}\right] + \mathbb{P}\{\neg E_t\} e^{\lambda t h} \\ &\stackrel{\text{Eq. 18}}{\leq} e^{\frac{\lambda^2 h^2}{8}} \mathbb{E}\left[e^{\lambda \sum_{i=1}^{t-1} (\mu - X_i)}\right] + c_p e^{t(\lambda h - c_e)} \\ &\stackrel{(*)}{\leq} e^{\frac{\lambda^2 h^2}{8} t} + \sum_{\tau=1}^t e^{\frac{\lambda^2 h^2}{8} (t-\tau)} \cdot c_p e^{\tau(\lambda h - c_e)} \\ &\text{by the auxiliary step in Eqs. 49-51 below} \\ &\leq e^{\frac{\lambda^2 h^2 t}{8}} \left[1 + c_p \sum_{\tau=1}^{\infty} e^{-\frac{\lambda^2 h^2 \tau}{8}} e^{\tau(\lambda h - c_e)}\right]. \end{aligned} \quad (48)$$

Considering the recursion

$$f(t) = \theta f(t-1) + g(t), \quad (49)$$

it is easy to verify that, for all  $0 \leq c < t$ ,

$$f(t) = \theta^{t-c} f(c) + \sum_{\tau=c+1}^t \theta^{t-\tau} g(\tau). \quad (50)$$

Given that, the bound  $(*)$  in Eq. 48 is obtained by setting

$$\begin{aligned} \theta &= e^{\frac{\lambda^2 h^2}{8}}, \\ f(t) &= \mathbb{E}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right], \\ g(t) &= c_p e^{t(\lambda h - c_e)}. \end{aligned} \quad (51)$$

Now, by Markov inequality, for any  $\lambda > 0$ ,

$$\begin{aligned}
& \mathbb{P} \left\{ \mu t - \sum_{i=1}^t X_i \geq t\delta \right\} \\
& \leq e^{-\lambda t\delta} \mathbb{E} \left[ e^{\lambda \sum_{i=1}^t (\mu - X_i)} \right] \\
& \stackrel{\text{Eq.48}}{\leq} e^{-\lambda t\delta} e^{\frac{\lambda^2 h^2 t}{8}} \left[ 1 + c_p \sum_{\tau=1}^{\infty} e^{-\frac{\lambda^2 h^2 \tau}{8}} e^{\tau(\lambda h - c_e)} \right] \\
& = e^{-\frac{2\delta^2 c_e}{h^2} t} e^{\frac{\delta^2 c_e^2}{2h^2} t} \left[ 1 + c_p \sum_{\tau=1}^{\infty} e^{-\frac{\delta^2 c_e^2}{2h^2} \tau} e^{\tau(\frac{2\delta c_e}{h} - c_e)} \right] \tag{52} \\
& \text{by setting } \lambda = \frac{2\delta c_e}{h^2} \\
& \leq e^{-\frac{3\delta^2 c_e}{2h^2} t} \left[ 1 + c_p \frac{2h^2}{\delta^2 c_e^2} e^{-\frac{\delta^2 c_e^2}{2h^2}} \right] \\
& \text{since } \delta < \frac{h}{2} \\
& \leq e^{-\frac{3\delta^2 c_e}{2h^2} t} \left[ 1 + c_p \frac{2h^2}{\delta^2 c_e^2} \right].
\end{aligned}$$

The second bound can be proven in much the same way. ■

**Discussion 14** *Note that the above bound was obtained for a particular choice of  $\lambda$  that maximizes the coefficient term in the exponent. Other choices of  $\lambda$  may result in a smaller coefficient in the exponent, but also a decreased leading constant. In particular, setting  $\lambda = \frac{2\delta c_e \beta}{h^2}$ , for any  $0 < \beta < 1$ , yields the following bound*

$$\begin{aligned}
& \mathbb{P} \left\{ \mu t - \sum_{i=1}^t X_i \geq t\delta \right\} \\
& \leq e^{-\frac{2\delta^2 c_e \beta}{h^2} t} e^{\frac{\delta^2 c_e^2 \beta^2}{2h^2} t} \left[ 1 + c_p \sum_{\tau=1}^{\infty} e^{-\frac{\delta^2 c_e^2 \beta^2}{2h^2} \tau} e^{\tau(\frac{2\delta c_e \beta}{h} - c_e)} \right] \tag{53} \\
& \leq e^{-\frac{3\delta^2 c_e \beta}{2h^2} t} \left[ 1 + \frac{c_p}{c_e (1 - \beta)} e^{-\frac{c_e (1 - \beta)}{2h^2}} \right].
\end{aligned}$$

#### Proof of Lemma 5:

Lemma 4 implies that, with probability approaching 1 exponentially fast, the state-space samples issued at a level with  $h + 1$  steps-to-go are optimal. That is, their expectation equals the actual  $Q$ -value. Therefore, by Lemma 4, we have

$$\mathbb{P} \{ \mathbb{E} [X_{t,h+1}(s, a) \mid X_{1,h+1}(s, a), \dots, X_{t-1,h+1}(s, a)] \neq Q(s, a) \} \leq c_p e^{-c_e t}, \tag{54}$$

where  $c_p = 2Kh(2 + c_h)$  and  $c_e = \frac{pc'_h}{6K}$ . It is also easy to see that  $0 \leq X_i \leq h + 1$ , and thus the conditions of Lemma 3 are satisfied. In turn, from Lemma 3 for  $\delta = \frac{d}{2}$  and random



variables with support  $[0, h + 1]$ ,

$$\begin{aligned}
& \mathbb{P} \left\{ \widehat{Q}_{h+1}(s, a) - Q_{h+1}(s, a) \geq \frac{d}{2} \mid n_{h+1}(s, a) = t \right\} \\
& \leq \left[ 1 + c_p \frac{2(h+1)^2}{\left(\frac{d}{2}\right)^2 c_e^2} \right] \cdot e^{-\frac{3\left(\frac{d}{2}\right)^2 c_e}{2(h+1)^2} t} \\
& \leq e^{-\frac{d^2 p c'_h}{16(h+1)^2 K} t} \left[ 1152 \cdot \frac{K^3 (h+1)^3 (2 + c_h)}{d^2 p^2 c_h'^2} \right],
\end{aligned} \tag{55}$$

and, similarly,

$$\begin{aligned}
& \mathbb{P} \left\{ \widehat{Q}_{h+1}(s, a) - Q_{h+1}(s, a) \leq -\frac{d}{2} \mid n_{h+1}(s, a) = t \right\} \\
& \leq e^{-\frac{d^2 p c'_h}{16(h+1)^2 K} t} \left[ 1152 \cdot \frac{K^3 (h+1)^3 (2 + c_h)}{d^2 p^2 c_h'^2} \right] \\
& \leq e^{-\frac{d^2 p c'_h}{16(h+1)^2 K} t} \left[ 3456 \cdot \frac{K^3 (h+1)^3 c_h}{d^2 p^2 c_h'^2} \right] \\
& \quad \text{since } 2 + c_h \leq 3c_h.
\end{aligned} \tag{56}$$

■

### Proof of Lemma 2, induction step:

Note that the proof of Lemma 5 is basically the proof of the induction step for the key part of Lemma 2, that is, Eq. 14. The only thing that remains to be finalized is the correctness of Eqs. 15 and 16 for  $h + 1$ , and these can be verified by substitution of  $c_h$  and  $c'_h$  in Eq. 56 by the respective expressions (for  $h$ ) from Eqs. 15 and 16. ■

### Proof of Theorem 1:

The proof for our main results follows by using the same techniques as above. Note that, by the Hoeffding inequality, after  $n > 0$  iterations of BRUE, for each action  $a \in A(s_0)$ , it holds that

$$\mathbb{P} \left\{ n_H(s_0, a) \leq \frac{n}{2KH} \right\} \leq e^{-\frac{1}{2K^2 H} n}. \tag{57}$$

Given that,

$$\begin{aligned}
& \mathbb{P} \left\{ \pi_n^B(s_0, H) \neq \pi^*(s_0, H) \right\} \\
& \leq \sum_{a \neq \pi^*(s_0, H)} \left[ \mathbb{P} \left\{ \widehat{Q}_H(s_0, a) \geq Q_H(s_0, a) + \frac{d}{2} \right\} + \right. \\
& \quad \left. \mathbb{P} \left\{ \widehat{Q}_H(s_0, \pi^*(s_0, H)) \leq Q_H(s_0, \pi^*(s_0, H)) - \frac{d}{2} \right\} \right].
\end{aligned} \tag{58}$$

For a sub-optimal action  $a$ ,

$$\begin{aligned}
& \mathbb{P} \left\{ \widehat{Q}_H(s_0, a) \geq Q_H(s_0, a) + \frac{d}{2} \right\} \\
&= \sum_{t=1}^{\frac{n}{H}} \mathbb{P} \left\{ \widehat{Q}_H(s_0, a) \geq Q_H(s_0, a) + \frac{d}{2} \mid n_H(s_0, a) = t \right\} \mathbb{P} \{n_H(s_0, a) = t\} \\
&\leq \mathbb{P} \left\{ n_H(s_0, a) \leq \frac{n}{2KH} \right\} \\
&\quad + \sum_{t=1+\frac{n}{2KH}}^{\frac{n}{H}} \mathbb{P} \left\{ \widehat{Q}_H(s_0, a) \geq Q_H(s_0, a) + \frac{d}{2} \mid n_H(s_0, a) = t \right\} \mathbb{P} \{n_H(s_0, a) = t\} \\
&\stackrel{\text{Lemma 2}}{\leq} e^{-\frac{1}{2K^2H}n} + \sum_{t=1+\frac{n}{2KH}}^{\frac{n}{H}} c_H e^{-c'_H t} \cdot \mathbb{P} \{n_H(s_0, a) = t\} \\
&\leq e^{-\frac{1}{2K^2H}n} + c_H e^{-\frac{c'_H}{2KH}n} \\
&\leq 2c_H e^{-\frac{c'_H}{2KH}n}.
\end{aligned} \tag{59}$$

Using exactly the same line of bounding, we obtain

$$\mathbb{P} \left\{ \widehat{Q}_H(s_0, \pi^*(s_0, H)) \leq Q_H(s_0, \pi^*(s_0, H)) - \frac{d}{2} \right\} \leq 2c_H e^{-\frac{c'_H}{2KH}n}, \tag{60}$$

and thus

$$\mathbb{P} \{ \pi_n^B(s_0, H) \neq \pi^*(s_0, H) \} \leq 4Kc_H e^{-\frac{c'_H}{2KH}n}. \tag{61}$$

Eqs. 11, 12, and 13 of Theorem 1 are then obtained by substitution of  $c_H$  and  $c'_H$  in Eq. 61 with the respective expressions from Eqs. 15 and 16. In turn, Eq. 10 of Theorem 1 stems from Eqs. 11, horizon  $H$ , and per-step rewards being in  $[0, 1]$ .  $\blacksquare$

## Appendix B. Proof of Theorem 6

We first prove the modified Hoeffding-Azuma inequality for partial sums.

### Proof of Lemma 8 (Modified Hoeffding-Azuma inequality for partial sums):

Let  $E_t$  be the event that  $\mathbb{E}[X_t \mid X_1, \dots, X_{t-1}] = \mu$ , and let

$$Y_t \triangleq X_t - \mu \mid X_1(\omega), \dots, X_{t-1}(\omega).$$

The random variable  $Y_t$  is bounded by  $h - \mu \geq Y_t \geq -\mu$ , and furthermore, for  $\omega \in E_t$ ,  $\mathbb{E}Y_t = 0$ . Therefore, using Fact 13, for all  $\omega \in E_t$  and  $\lambda \in \mathbb{R}^+$ , it holds that

$$\mathbb{E}[e^{\lambda Y}] \leq e^{\frac{\lambda^2 h^2}{8}}. \quad (62)$$

Moreover,

$$\begin{aligned} & \mathbb{E}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)}\right] \\ &= \mathbb{E}_{E_t}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)}\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)}\right] \\ &= \mathbb{E}_{E_t}\left[\mathbb{E}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)} \mid X_1, \dots, X_{t-1}\right]\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)}\right] \\ &= \mathbb{E}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^{t-1} (\mu - X_i)} \cdot \mathbb{E}\left[e^{\lambda (\mu - X_t)} \mid X_1, \dots, X_{t-1}\right]\right] + \mathbb{E}_{\neg E_t}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)}\right] \\ &\leq e^{\frac{\lambda^2 h^2}{8}} \mathbb{E}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^{t-1} (\mu - X_i)}\right] + \mathbb{P}\{\neg E_t\} e^{\lambda h \lceil \alpha t \rceil} \\ &\stackrel{\text{Eq. 18}}{\leq} e^{\frac{\lambda^2 h^2}{8}} \mathbb{E}\left[e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^{t-1} (\mu - X_i)}\right] + c_p e^{\lambda h \lceil \alpha t \rceil - c_e t} \\ &\stackrel{(*)}{\leq} e^{\frac{\lambda^2 h^2}{8} \lceil \alpha t \rceil} + \sum_{\tau=t-\lceil \alpha t \rceil}^t e^{\frac{\lambda^2 h^2}{8}(t-\tau)} \cdot c_p e^{\lambda h \lceil \alpha \tau \rceil - c_e \tau} \end{aligned} \quad (63)$$

by the auxiliary step in Eqs. 49-66 below.

Considering the recursion

$$f(t) = \theta f(t-1) + g(t), \quad (64)$$

it is easy to verify that, for all  $0 \leq c < t$ ,

$$f(t) = \theta^{t-c} f(c) + \sum_{\tau=c+1}^t \theta^{t-\tau} g(\tau). \quad (65)$$

Given that, the bound  $(*)$  in Eq. 48 is obtained by setting

$$\begin{aligned} \theta &= e^{\frac{\lambda^2 h^2}{8}}, \\ f(t) &= \mathbb{E}\left[e^{\lambda \sum_{i=1}^t (\mu - X_i)}\right], \\ g(t) &= c_p e^{t(\lambda h - c_e)}. \end{aligned} \quad (66)$$

Now, by Markov inequality, for any  $\lambda > 0$ ,

$$\begin{aligned}
& \mathbb{P} \left\{ \mu \lceil \alpha t \rceil - \sum_{i=t-\lceil \alpha t \rceil}^t X_i \geq \lceil \alpha t \rceil \delta \right\} \\
& \leq e^{-\lambda \delta \lceil \alpha t \rceil} \mathbb{E} \left[ e^{\lambda \sum_{i=t-\lceil \alpha t \rceil}^t (\mu - X_i)} \right] \\
& \stackrel{\text{Eq. 48}}{\leq} e^{-\lambda \delta \lceil \alpha t \rceil} \left[ e^{\frac{\lambda^2 h^2}{8} \lceil \alpha t \rceil} + \sum_{\tau=t-\lceil \alpha t \rceil}^t e^{\frac{\lambda^2 h^2}{8} (t-\tau)} \cdot c_p e^{\lambda h \lceil \alpha \tau \rceil - c_e \tau} \right] \\
& = e^{-\frac{2\delta^2 c_e}{h^2} \lceil \alpha t \rceil} \left[ e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} + \sum_{\tau=t-\lceil \alpha t \rceil}^t e^{\frac{\delta^2 c_e^2}{2h^2} (t-\tau)} \cdot c_p e^{\frac{2\delta c_e}{h} \lceil \alpha \tau \rceil - c_e \tau} \right] \\
& \text{by setting } \lambda = \frac{2\delta c_e}{h^2} \\
& \leq e^{-\frac{2\delta^2 c_e}{h^2} \lceil \alpha t \rceil} \left[ e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} + e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} \sum_{\tau=t-\lceil \alpha t \rceil}^t c_p e^{\frac{2\delta c_e}{h} \lceil \alpha \tau \rceil - c_e \tau} \right] \tag{67} \\
& \leq e^{-\frac{2\delta^2 c_e}{h^2} \lceil \alpha t \rceil} e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} \left[ 1 + c_p \sum_{\tau=t-\lceil \alpha t \rceil}^t e^{c_e \lceil \alpha \tau \rceil - c_e \tau} \right] \\
& \text{since } \delta < \frac{h}{2} \\
& \leq e^{-\frac{2\delta^2 c_e}{h^2} \lceil \alpha t \rceil} e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} \left[ 1 + c_p \sum_{\tau=t-\lceil \alpha t \rceil}^t e^{-c_e (1-\alpha) \tau} \right] \\
& \leq e^{-\frac{2\delta^2 c_e}{h^2} \lceil \alpha t \rceil} e^{\frac{\delta^2 c_e^2}{2h^2} \lceil \alpha t \rceil} \left[ 1 + \frac{c_p}{c_e (1-\alpha)} e^{-c_e (1-\alpha)^2 t} \right] \\
& \leq e^{-\frac{3\delta^2 c_e}{2h^2} \alpha t} \left[ 1 + \frac{c_p}{c_e (1-\alpha)} e^{-c_e (1-\alpha)^2 t} \right].
\end{aligned}$$

■

Obviously, at leaf nodes there is no point in choosing  $\alpha < 1$  since there is no bias. Therefore, for  $h = 1$  we can use the same constants  $c_1 = 1$  and  $c'_1 = \frac{p^H}{K^H}$ . Since  $c'_h$  is decreasing with  $h$ , we have  $c'_h \leq \frac{p^{H-h}}{K^{H-h}}$  for all  $1 \leq h \leq H$ , and thus Lemma 4 is valid. Lemma 5 relies on the modified Hoeffding-Azuma inequality, which is no longer valid in the context of  $\text{BRUE}(\alpha)$ . Instead, we apply its modification, Lemma 8 for partial sums, to prove the induction step

$$\begin{aligned}
& \mathbb{P} \left\{ \widehat{Q}_{h+1}(s, a) - Q_{h+1}(s, a) \geq \frac{d}{2} \mid n_{h+1}(s, a) = t \right\} \\
& \leq e^{-\frac{3d^2 p c'_h}{48 K h^2} \alpha t} \left[ 1 + \frac{12 K^2 h (2 + c_h)}{p c'_h (1 - \alpha)} e^{-\frac{p c'_h (1 - \alpha)^2}{6 K} t} \right] \tag{68}
\end{aligned}$$

and, similarly,

$$\begin{aligned} & \mathbb{P} \left\{ \widehat{Q}_{h+1}(s, a) - Q_{h+1}(s, a) \leq -\frac{d}{2} \mid n_{h+1}(s, a) = t \right\} \\ & \leq e^{-\frac{3d^2 pc'_h}{48Kh^2} \alpha t} \left[ 1 + \frac{12K^2 h(2 + c_h)}{pc'_h(1 - \alpha)} e^{-\frac{pc'_h(1-\alpha)^2}{6K} t} \right]. \end{aligned} \quad (69)$$

The induction step is satisfied, e.g., with  $c'_{h+1} = \min \left\{ \frac{3d^2 pc'_h \alpha}{48Kh^2}, \frac{pc'_h(1-\alpha)^2}{6K} \right\}$  and  $c_{h+1} = 1 + \frac{12K^2 h(2+c_h)}{pc'_h(1-\alpha)}$ .

Since  $c_h$  is increasing in  $h$  and  $c'_h$  is decreasing in  $h$ , the term  $\frac{12K^2 h(2+c_h)}{pc'_h(1-\alpha)}$  also increases in  $h$ . The larger the constant grows, the more beneficial it might be to increase the exponent coefficient that multiplies that constant by decreasing  $\alpha$  at the expense of decreasing the exponent coefficient that multiplies 1. Clearly, the tradeoff depends also on  $t$ , the number of samples of action  $a$  in node  $(s, h)$ . Therefore, as  $h$  increases, smaller values of  $\alpha$  would be more appealing.

# Deterministic Oversubscription Planning as Heuristic Search: Abstractions and Reformulations

**Carmel Domshlak**

**Vitaly Mirkis**

*Faculty of Industrial Engineering & Management,  
Technion - Israel Institute of Technology,  
Haifa, Israel*

DCARMEL@IE.TECHNION.AC.IL

MIRKIS@TX.TECHNION.AC.IL

## Abstract

In deterministic oversubscription planning (OSP), the objective is to achieve an as valuable as possible subset of goals within a fixed allowance of the total action cost (Smith, 2004). Although numerous applications in various fields share this objective, no substantial algorithmic advances have been made in deterministic OSP, and this in contrast to a tremendous progress that has been achieved in the area of classical deterministic planning (Russell & Norvig, 2009). Tracing the key sources of progress in classical planning, we identify a severe lack of effective domain-independent approximations for OSP.

With our focus here on optimal planning, two classes of approximation techniques have been found especially useful in the context of optimal classical planning: those based on state-space *abstractions* and these based on logical *landmarks* for goal reachability. The question we study here is whether some similar-in-spirit, yet possibly mathematically different, approximation techniques can be developed for OSP. In the context of abstractions, we define the notion of additive abstractions for OSP, study the complexity of deriving effective abstractions from a rich space of hypotheses, and reveal some substantial, empirically relevant islands of tractability. In the context of landmarks, we show how standard goal-reachability landmarks of certain classical planning tasks can be compiled into the OSP task of interest, resulting in an equivalent OSP task with a lower cost allowance, and thus with a smaller search space. Our empirical evaluation confirms the effectiveness of the proposed techniques, and opens a wide gate for further developments in oversubscription planning.

## 1. Introduction

The tools of automated action planning are developed to allow autonomous systems selecting a course of action “to make things done”. Deterministic planning is probably the most basic, and thus the most fundamental, setting of automated action planning (Russell & Norvig, 2009). At a high level, deterministic planning is a problem of finding trajectories of interest in large-scale yet concisely represented state-transition systems. Computational approaches to deterministic planning vary around the way those “trajectories of interest” are defined.

The basic structure of acting in situations with underconstrained or overconstrained resources is respectively captured by what these days is called “classical” deterministic planning (Fikes & Nilsson, 1971), and in what Smith (2004) baptized as “oversubscription” deterministic planning (OSP). In classical planning, the task is to find an as *cost-effective*

trajectory as possible to a *goal-satisfying* state. In oversubscription planning, the task is to find an as *goal-effective* (or *valuable*) state as possible via a *cost-satisfying* trajectory. In optimal classical planning and in optimal OSP, the tasks are further constrained to finding only *most* cost-effective trajectories and *most* goal-effective states, respectively. Together, classical planning and OSP constitute the most fundamental variants of deterministic planning, with many other variants of deterministic planning, such as net-benefit planning and cost-bounded planning, being defined in terms of mixing and relaxing the two.<sup>1</sup>

While OSP has been advocated over the years on par with classical planning, so far, the theory and practice of classical planning have been studied and advanced much more intensively. The remarkable success and continuing progress of heuristic-search solvers for classical planning is one notable example. Primary enablers of this success are the advances in domain-independent approximations, or heuristics, of the cost needed to achieve a goal state from a given state. It is thus possible that having a similarly rich palette of effective heuristic functions for OSP would advance the state-of-the-art in that problem.

With our focus here on optimal planning, two classes of approximation techniques have been found especially useful in the context of optimal classical planning: those based on state-space *abstractions* (Edelkamp, 2001; Haslum, Botea, Helmert, Bonet, & Koenig, 2007; Helmert, Haslum, & Hoffmann, 2007; Katz & Domshlak, 2010a) and these based on logical *landmarks* for goal reachability (Karpas & Domshlak, 2009; Helmert & Domshlak, 2009; Domshlak, Katz, & Lefler, 2012; Bonet & Helmert, 2010a; Pommerening & Helmert, 2013). Considering OSP as heuristic search, a question is then whether some similar-in-spirit, yet possibly mathematically different, approximation techniques can be developed for heuristic-search OSP. This is precisely the question we study here.

- Starting with the most basic question of what state-space abstractions for OSP actually are, we show that the very notion of abstraction substantially differs between classical planning and OSP. Hence, first we define (additive) abstractions and abstraction heuristics for OSP. We then investigate computational complexity of deriving effective abstraction heuristics in the scope of homomorphic abstraction skeletons, paired with cost, value, and budget partitions. Along with revealing some significant islands of tractability, this study exposes an interesting interplay between knapsack-style problems of combinatorial optimization, continuous convex optimization, and certain principles borrowed from explicit abstractions for classical planning.
- We introduce and study  $\varepsilon$ -landmarks, the logical properties of OSP plans that achieve valuable states. We show that  $\varepsilon$ -landmarks correspond to regular goal-reachability landmarks of certain classical planning tasks that can be straightforwardly derived from the OSP tasks of interest. We then show how such  $\varepsilon$ -landmarks can be compiled back into the OSP task of interest, resulting in an equivalent OSP task, but with a stricter cost satisfaction constraint, and thus with a smaller effective search space. Finally, we show how such landmark-based task enrichment can be combined in a mutually stratifying way with the *BFBB* search used for OSP planning, resulting in an incremental procedure that interleaves search and landmark discovery. The

---

1. The connections and differences between some popular setups of deterministic planning are discussed in Section 2.

entire framework is independent of the OSP planner specifics, and in particular, of the heuristic functions it employs.

Our empirical evaluation on a large set of OSP tasks confirms the effectiveness of the proposed techniques. Also, to our knowledge, our implementation constitutes the first domain-independent solver for optimal OSP, and we hope that more advances in this important computational problem will follow.

This work is a revision and extension of the formulations and results presented by the authors at ICAPS-2013 and ECAI-2014 (Mirkis & Domshlak, 2013, 2014). The paper is structured as follows. In Section 2 we formulate a general model of deterministic planning, define several variants of deterministic planning in terms of this model, and, in particular, show that oversubscription planning differ conceptually not only from classical planning, but also from other popular setups of deterministic planning such as net-benefit planning and cost-bounded planning. In Section 2 we also specify a simple model representation language for OSP, as well as provide the essential background on heuristic search, and, in particular, on OSP as heuristic search. Sections 3 and 4 are devoted to abstractions and abstraction approximations for OSP, respectively. Section 5 is devoted to exploiting reachability landmarks in OSP tasks. In Section 6 we conclude and discuss some promising directions for future work. The paper ends with two appendices: For the sake of readability, some of the proofs are delegated to Appendix A, and some details of the empirical results are delegated to Appendix B.

## 2. Background

As we already mentioned in the introduction, specific variants of deterministic planning differ in the way the interest and preference over trajectories are defined. For instance, in “classical planning” (Fikes & Nilsson, 1971), a trajectory is of interest if it connects a designated initial state to one of the designated goal states, with the preference being towards trajectories with lower total cost of the transitions along them. Among other, “non-classical” variants of deterministic planning are

- oversubscription planning (Smith, 2004), the topic of our interest here, as well as
- net-benefit planning (Sanchez & Kambhampati, 2005; Baier, Bacchus, & McIlraith, 2007; Bonet & Geffner, 2008; Benton, Do, & Kambhampati, 2009; Coles & Coles, 2011; Keyder & Geffner, 2009),
- cost-bounded (also known as resource-constrained) planning (Haslum & Geffner, 2001; Hoffmann, Gomes, Selman, & Kautz, 2007; Gerevini, Saetti, & Serina, 2008; Thayer & Ruml, 2011; Thayer, Stern, Felner, & Ruml, 2012; Haslum, 2013; Nakhost, Hoffmann, & Müller, 2012), and
- planning with preferences over temporal properties of the trajectories (Baier et al., 2007; Baier, Bacchus, & McIlraith, 2009; Gerevini, Haslum, Long, Saetti, & Dimopoulos, 2009; Benton, Coles, & Coles, 2012).



Interestingly, while working on this paper, we have learned that quite a few different variants of deterministic planning are often collectively referred to as “oversubscription planning”. As a result, the difference between them in terms of expressiveness is not necessarily clear, and thus, the positioning of what we do here with respect to what has already been done in the collective sense of “oversubscription planning” is not always apparent. This is the issue we are going to address first.

## 2.1 Models

Adopting and extending the notation of Geffner and Bonet (2013), many variants of deterministic planning, including classical planning, as well as many popular non-classical variants, can be seen as special cases of a state model

$$M = \langle S, s_0, u, O, \varphi, c, Q \rangle \quad (1)$$

with:

- a finite and discrete state space  $S$ ,
- an initial state  $s_0 \in S$ ,
- a state value function  $u : S \mapsto \mathbb{R}^{0+} \cup \{-\infty\}$ ,
- operators  $O(s) \subseteq O$  applicable in each state  $s \in S$ ,
- a deterministic state transition function  $\varphi(s, o)$  such that  $s' = \varphi(s, o)$  stands for the state resulting from applying  $o \in O(s)$  in  $s$ .
- an operator cost function  $c : O \rightarrow \mathbb{R}^{0+}$ , and
- a *quality* measure  $Q : \mathcal{P} \mapsto \mathbb{R} \cup \{-\infty\}$ , where  $\mathcal{P}$  is the (infinite) set of trajectories from  $s_0$  along operators  $O$ .

In this model, *any* trajectory  $\pi \in \mathcal{P}$  is a solution, with preference being towards the solutions of higher quality. In what follows,  $s[\pi]$  stands for the the end-state of a trajectory  $\pi$  applied at state  $s$ , and  $c(\pi) = \sum_{o \in \pi} c(o)$  is the additive cost of  $\pi$ . Likewise, by *graphical skeleton*  $G_M = \langle S, T_\varphi, O \rangle$  of a model  $M$  we refer to the edge-annotated, *unweighted* digraph induced by  $M$  naturally as follows: The nodes of  $G_M$  are the states  $S$ , the edge labels are the operators  $O$ , and  $T_\varphi$  contains an edge from  $s$  to  $s'$  labeled with  $o$  iff  $o \in O(s)$  and  $s' = \varphi(s, o)$ .

First, consider a quality measure

$$Q^+(\pi) = u(s[\pi]) - c(\pi). \quad (2)$$

This measure assumes that state values and operator costs are comparable, and thus trades between the value of the end-state and the cost of the trajectory. Consider now a fragment of the state model (1), instances of which all have the quality measure  $Q^+$ , and for each instance, the value function

$$u(s) = \begin{cases} \varepsilon, & s \in S_{goal} \\ -\infty, & \text{otherwise} \end{cases} \quad (3)$$

		ACTION COST	
		preference	constraint
END-STATE VALUE	preference	Net Benefit	Oversubscription
	constraint	Classical	Cost-bounded

Figure 1: Schematic classification of four deterministic planning models along the strictness with which they approach the cost of operator sequences and the value of the operator sequence end-states. White blocks are for planning models that can be solved as single-source single-target shortest path problems.

partitions the state space into  $S_{goal} \subseteq S$ , on which  $u$  takes a finite value  $\varepsilon \geq 0$ , and the rest of the states, on which  $u$  takes the value of  $-\infty$ . Finding an optimal solution for an instance  $M$  of this fragment corresponds to finding a *shortest path* from  $s_0$  to a *single* node  $s_*$  in an edge-weighted digraph  $G$ , which is obtained from  $G_M$  by (i) annotating the edges of the latter with costs  $c$ , and (ii) adding a dummy node  $s_*$  and zero-cost edges from all goal nodes  $s \in S_{goal}$  to  $s_*$ . While specified in a non-canonical way, it is not hard to verify that this fragment corresponds to the model of *classical planning*, with  $S_{goal}$  being the so called goal states.

Staying with the quality measure  $Q^+$  and removing now the requirement on  $u$  to comply with Eq. 3, we obtain a fragment that generalizes classical planning, and constitutes the basic model of what is called *net-benefit planning* (Sanchez & Kambhampati, 2005). Importantly, as it was noticed by Keyder and Geffner (2009) (in technically different, yet semantically similar terms), any instance  $M$  of this fragment can be reduced to finding a shortest path from a single node  $s_0$  to a *single* node  $s_*$  in an edge-weighted digraph  $G$ , obtained from  $G_M$  by (i) annotating edges of  $G_M$  with costs  $c$ , (ii) adding a dummy node  $s_*$  and edges from all nodes  $s \in S$  to  $s_*$ , and (ii) setting the cost of each such new edge  $(s, s_*)$  to  $\sum_{s' \in S \setminus \{s\}} u(s')$ . In particular, this equivalence between classical and basic net-benefit planning at the level of the computational model allowed Keyder and Geffner (2009) to show how certain standard representation formalisms for net-benefit planning can be efficiently compiled to a standard classical planning formalism.

Consider now an alternative quality measure

$$Q^b(\pi) = \begin{cases} u(s[\pi]), & c(\pi) \leq b \\ -\infty, & \text{otherwise} \end{cases}, \quad (4)$$

where  $b \in \mathbb{R}^{0+}$  is a predefined bound on the cost of the trajectories. The fragment of the basic model, instances of which are characterized by having the quality measure  $Q^b$  and the “ $\varepsilon$  or  $-\infty$ ” value functions as in Eq. 3, constitutes the model of what is called *cost-bounded planning* (Thayer & Ruml, 2011). Here as well, finding an optimal solution for a problem instance  $M$  corresponds to finding a shortest path from  $s_0$  to  $s_*$  in an edge-weighted digraph  $G$ , which is derived from  $G_M$  identically to the case of classical planning.<sup>2</sup> This, in particular, explains why it is only natural for heuristic-search methods for cost-bounded planning to exploit heuristics developed for classical planning (Haslum, 2013).

We now arrive to a forth fragment of the basic model. Staying with the quality measure  $Q^b$  and removing the requirement on  $u$  to to comply with Eq. 3, we obtain a fragment that generalizes cost-bounded planning, and constitutes the model of *oversubscription planning* (Smith, 2004). As it is illustrated by Figure 1, the hard constraint of classical planning translates to soft preference in OSP, and the hard constraint of OSP translates to soft preference in classical planning. However, in contrast to cost-optimal, net-benefit, and classical planning, this fragment does not appear to be reducible to the single-source single-target shortest path problem. In terms of the digraph  $G$  obtained from  $G_M$  by annotating the edges with costs  $c$ , finding an optimal solution to an instance of oversubscription planning requires (i) finding shortest paths from  $s_0$  to *all states*  $s \in S$  with  $u(s) > 0$ , (ii) filtering out from these states those are not reachable from  $s_0$  within the cost allowance  $b$ , and (iii) selecting from the remaining states a state that maximises  $u$ .

This contrast between oversubscription planning and all the three other popular variants of deterministic planning discussed above has at least two important implications. First, while a single shortest path can be searched for using best-first forward search procedures such as  $A^*$ , searching for shortest paths to numerous targets simultaneously requires a different, more exhaustive, forward search framework such as branch-and-bound. Second, net-benefit and cost-bounded planning clearly have the potential to (directly or indirectly) reuse the rich toolbox of heuristic functions that have been developed over the years for classical planning. In contrast, due to the differences in the underlying computational model, the same is not necessarily true for oversubscription planning, and examining this issue is precisely the focus of our work here.

## 2.2 Notation

For  $k \in \mathbb{N}^+$ , by  $[k]$  we denote the set  $\{1, 2, \dots, k\}$ . By  $||x||$  we refer to the representation size of object  $x$ , not to be confused with  $|x|$ , which denotes the number of elements in set  $x$ . An assignment of a variable  $v$  to value  $d$  is denoted by  $\langle v/d \rangle$ ; we often refer to such single variable assignments as *propositions*.

## 2.3 Model Representation

Departing from a very general model of oversubscription planning, in what follows we restrict our attention to instances of that model that are compactly representable in a language close to the  $SAS^+$  language for classical planning (Bäckström & Klein, 1991; Bäckström &

---

2. To be entirely precise here, once a shortest path  $\pi$  from  $s_0$  to  $s_*$  is found, it should still be checked against the cost bound  $b$ . This test, however, is local to  $\pi$ , and problem solving finishes independently of the test’s outcome.

Nebel, 1995). In this language, a deterministic **oversubscription planning (OSP)** task is given by a sextuple

$$\langle V, s_0, u; O, c, b \rangle, \quad (5)$$

where

- (1)  $V = \{v_1, \dots, v_n\}$  is a finite set of finite-domain *state variables*, with each complete assignment to  $V$  representing a *state*, and  $S = \text{dom}(v_1) \times \dots \times \text{dom}(v_n)$  being the *state space* of the task;
- (2)  $s_0 \in S$  is a designated *initial state*;
- (3)  $u$  is an efficiently computable *state value* function  $u : S \rightarrow \mathbb{R}^{0+}$ ;
- (4)  $O$  is a finite set of *operators*, with each operator  $o \in O$  being represented by a pair  $\langle \text{pre}(o), \text{eff}(o) \rangle$  of partial assignments to  $V$ , called *preconditions* and *effects* of  $o$ , respectively;
- (5)  $c_i : O \rightarrow \mathbb{R}^{0+}$  is an *operator cost* function;
- (6)  $b \in \mathbb{R}^{0+}$  is a *cost budget* allowed for the task.

Mapping a task description in this language into our basic model, an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$  induces the model  $M_\Pi = \langle S, s_0, u, O, \varphi, c, Q^b \rangle$ , with  $Q^b$  being the quality measure (4) instantiated with the  $\Pi$ 's budget  $b$ , and the transition function  $\varphi$  being specified as follows. Let  $\mathcal{V}(p) \subseteq V$  denote the subset of variables instantiated by a partial assignment  $p$ . Similarly to the classical planning semantics of  $\text{SAS}^+$ , operator  $o$  is applicable in a state  $s$  iff  $s[v] = \text{pre}(o)[v]$  for all  $v \in \mathcal{V}(\text{pre}(o))$ . Applying  $o$  changes the value of each  $v \in \mathcal{V}(\text{eff}(o))$  to  $\text{eff}(o)[v]$ , and the resulting state is denoted by  $s[o]$ . This notation is only defined if  $o$  is applicable in  $s$ . Applying a sequence of operators  $\langle o_1, \dots, o_m \rangle$  to a state  $s$  is defined inductively as  $s[\epsilon] := s$  and  $s[o_1, \dots, o_j] := s[o_1, \dots, o_{j-1}][o_j]$ . An operator sequence  $\pi$  is called an *s-plan* if it is applicable in state  $s$  and  $Q^b(\pi) \neq -\infty$ , that is,  $c(\pi) \leq b$ .

For an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , by  $\mathcal{D} = \bigcup_{v \in V} \text{dom}(v)$  we denote the union of the (uniquely labeled) state-variable domains. For a state  $s$  and a proposition  $\langle v/d \rangle \in \mathcal{D}$ ,  $\langle v/d \rangle \in s$  is used as a shortcut notation for  $s[v] = d$ .

## 2.4 OSP as Heuristic Search

The two major ingredients of any heuristic-search planner are its search algorithm and heuristic function. In classical planning, the heuristic is typically a function  $h : S \rightarrow \mathbb{R}^{0+} \cup \{\infty\}$ , with  $h(s)$  estimating the cost  $h^*(s)$  of optimal  $s$ -plans. A heuristic  $h$  is **admissible** if it is *lower-bounding*, that is,  $h(s) \leq h^*(s)$  for all states  $s$ . All common heuristic search algorithms for optimal classical planning, such as  $A^*$ , require admissible heuristics.

In contrast, a heuristic in OSP is a function  $h : S \times \mathbb{R}^{0+} \rightarrow \mathbb{R}^{0+}$ , with  $h(s, b)$  estimating the value  $h^*(s, b)$  of optimal  $s$ -plans under cost budget  $b$ . A heuristic  $h$  is **admissible** if it is *upper-bounding*, that is,  $h(s, b) \geq h^*(s, b)$  for all states  $s$  and all cost budgets  $b$ . Here

```

BFBB ( $\Pi = \langle V, s_0, u; O, c, b \rangle$ )
  open := new max-heap ordered by  $f(n) = h(s[n], b - g(n))$ 
  initialize best solution  $n^* := \text{make-root-node}(s_0)$ 
  open.insert(make-root-node( $n^*$ ))
  closed :=  $\emptyset$ ;
  best-cost :=  $\emptyset$ 
  while not open.empty()
     $n := \text{open.pop-max}()$ 
    if  $f(n) \leq u(s[n^*])$ : break
    if  $s[n] \notin \text{closed}$  or  $g(n) < \text{best-cost}(s[n])$ :
      closed := closed  $\cup \{s[n]\}$ 
      best-cost( $s[n]$ ) :=  $g(n)$ 
      foreach  $o \in O(s[n])$ :
         $n' := \text{make-node}(s[n][o])$ 
        if  $g(n') > b$  or  $f(n') \leq u(s[n^*])$ : continue
        if  $u(s[n']) > u(s[n^*])$ : update  $n^* := n'$ 
        open.insert( $n'$ )
  return  $n^*$ 

```

Figure 2: Best-first branch-and-bound (*BFBB*) search for OSP

as well, search algorithms for optimal OSP, such as best-first branch-and-bound (*BFBB*),<sup>3</sup> require admissible heuristics, and this for pruning search branches without violating solution optimality.

Figure 2 depicts a pseudo-code description of *BFBB* for OSP.  $s[n]$  there denotes the state associated with search node  $n$ . Unlike in  $A^*$ , the order in which the nodes are selected from the OPEN list does not affect the optimality guarantees (though, of course, may seriously affect the empirical efficiency of the search). In Figure 2, the ordering of OPEN corresponds to the decreasing order of  $h(s[n], b - g(n))$ . The duplicate detection and reopening mechanisms in *BFBB* are similar to those in  $A^*$  (Pearl, 1984). In addition, *BFBB* maintains the best solution  $n^*$  found so far and uses it to prune all generated nodes evaluated no higher than  $u(s[n^*])$ . Likewise, complying with the semantics of OSP, all generated nodes  $n$  with cost-so-far  $g(n)$  higher than the problem’s budget  $b$  are also immediately pruned. When the OPEN list becomes empty or the node  $n$  selected from the list promises less than the lower bound, *BFBB* returns (the plan associated with) the best solution  $n^*$ . If  $h$  is admissible, that is, the  $h$ -based pruning of the generated nodes is sound, then the returned plan is guaranteed to be optimal.

Returning now to the heuristic functions, in domain-independent planning they should be automatically derived from the description of the model in the language of choice. A useful heuristic function must be both efficiently computable from the description of the model, as well as relatively accurate in its estimates. Improving the accuracy of a heuristic

3. *BFBB* is also extensively used for net-benefit planning (Benton, van den Briel, & Kambhampati, 2007; Coles & Coles, 2011; Do, Benton, van den Briel, & Kambhampati, 2007), as well as some other variants of deterministic planning (Bonet & Geffner, 2008; Brafman & Chernyavsky, 2005).

function without substantially worsening the time complexity of computing it translates into faster search for plans.

In classical planning, numerous approximation techniques, such as monotonic relaxation (Bonet & Geffner, 2001, 2001; Hoffmann & Nebel, 2001), critical trees (Haslum & Geffner, 2000), network flow (van den Briel, Benton, Kambhampati, & Vossen, 2007; Bonet, 2013), logical landmarks for goal reachability (Richter, Helmert, & Westphal, 2008; Karpas & Domshlak, 2009; Helmert & Domshlak, 2009; Bonet & Helmert, 2010a), and abstractions (Edelkamp, 2001; Helmert et al., 2007; Katz & Domshlak, 2010a), have been translated to effective heuristic functions. Likewise, different heuristics for classical planning can also be combined into their point-wise maximizing and/or additive ensembles (Edelkamp, 2001; Haslum, Bonet, & Geffner, 2005; Coles, Fox, Long, & Smith, 2008; Katz & Domshlak, 2010b; Helmert & Domshlak, 2009).

In contrast, development of heuristic functions for OSP has not progressed beyond the initial ideas of Smith (2004). In principle, the reduction of Keyder and Geffner (2009) from net-benefit to classical planning can be used to reduce OSP to classical planning with *real-valued* state variables (Koehler, 1998; Helmert, 2002; Fox & Long, 2003; Hoffmann, 2003; Gerevini, Saetti, & Serina, 2003; Gerevini et al., 2008; Edelkamp, 2003; Dvorak & Barták, 2010; Coles, Coles, Fox, & Long, 2013). So far, however, progress in heuristic-search classical planning with numeric state variables has mostly been achieved around direct extensions of delete relaxation heuristics via “numeric relaxed planning graphs” (Hoffmann, 2003; Edelkamp, 2003; Gerevini et al., 2003, 2008). Unfortunately, these heuristics do not preserve information on consumable resources such as budgeted operator cost in oversubscription planning: The “negative” action effects that decrease the values of numeric variables are ignored, possibly up to some special handling of so-called “cyclic resource transfer” (Coles et al., 2013).

Approaching the deficit in effective heuristics for OSP, in the next section we study *abstractions for OSP*, from their very definition and properties, to the prospects of deriving admissible abstraction heuristics. In Section 5 we then study the prospects of adapting to OSP the toolbox of logical landmarks for goal reachability. To date, abstractions and landmarks are responsible for most state-of-the-art admissible heuristics for classical planning, and thus are of our special interest here.

### 3. Abstractions

The term “abstraction” is usually associated with simplifying the original model, factoring out details less crucial in the given context. Which details can be reduced and which should better be preserved, as well as how the abstraction is created and used, depends largely on the context (Cousot & Cousot, 1992; Clarke, Grumberg, & Peled, 1999; Helmert et al., 2007; Domshlak, Hoffmann, & Sabharwal, 2009; Katz & Domshlak, 2010b). In general terms, abstracting a model  $M$  corresponds to associating it with a set of (typically computationally more attractive) models  $M_1, \dots, M_k$  such that solutions to these models satisfy certain properties with respect to the solutions of  $M$ . In particular, in deterministic planning as heuristic search, abstractions are used to derive heuristic estimates for the states of the model of interest  $M$ : Given a state  $s$  of  $M$  and an abstraction  $M_1, \dots, M_k$ ,

- (1)  $s$  is mapped to some “abstract states”  $s_1 \in M_1, \dots, s_k \in M_k$ ,

- (2) the  $k$  models of the abstraction are solved for the respective initial states  $s_1, \dots, s_k$ , and
- (3) an aggregation of the quality of the resulting  $k$  solutions is used as the heuristic estimate for  $s$ .

Sometimes schematically and sometimes precisely, the process of constructing abstractions as above for a state model  $M = \langle S, s_0, u, O, \varphi, c, Q \rangle$  can be seen as a two-step process of

- (1) Selecting an **abstraction skeleton**  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$ , where each pair  $(G_i, \alpha_i)$  comprises an edge-labeled digraph  $G_i = \langle S_i, T_i, O_i \rangle$ , with nodes  $S_i$ , edges  $T_i$ , and edge labels  $O_i$ , and a state mapping  $\alpha_i : S \rightarrow S_i$ .
- (2) Extending  $\mathcal{AS}$  to a set of **abstract models**  $\mathcal{M} = \{M_1, \dots, M_k\}$ , such that, for  $i \in [k]$ ,  $G_i$  is the graphical skeleton  $G_{M_i}$  of  $M_i$ .

To be qualified as a valid abstraction of the model  $M$ , the resulting set of abstract models  $\mathcal{M}$  should satisfy certain conditions specific to the variant of the deterministic planning under consideration. For instance, the optimal solutions of abstract models in classical planning are required to be at most as costly as the respective solutions in the original models, with that constraint to be satisfied by individual abstract models in case of max-aggregation (Pearl, 1984), or by the  $k$  abstract models jointly, in case of additive abstractions (Yang, Culberson, Holte, Zahavi, & Felner, 2008; Katz & Domshlak, 2010b). As we now show, the concept of abstractions in general, and additive abstractions in particular, is very different in OSP, and, for better and for worse, has many more degrees of freedom than the respective concepts in classical planning.

### 3.1 Abstractions of OSP Problems

Given an abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  for an OSP state model  $M = \langle S, s_0, u, O, \varphi, c, Q^b \rangle$ , each digraph  $G_i = \langle S_i, T_i, O_i \rangle$  implicitly defines a set of OSP state models consistent with it. This set is given by  $C_i \times U_i \times B_i$  where  $C_i$  is the set of all functions from operators  $O_i$  to  $\mathbb{R}^{0+}$ ,  $U_i$  is the set of all functions from states  $S_i$  to  $\mathbb{R}^{0+}$ , and  $B_i = \mathbb{R}^{0+}$ . In these terms, each point  $(c, u, b) \in C_i \times U_i \times B_i$  induces an OSP model consistent with  $G_i$ , and vice versa.

Connecting between these sets of models for all the digraphs in  $\mathcal{AS}$ , let

$$\begin{aligned} \mathbf{C} &= C_1 \times \dots \times C_k, \\ \mathbf{U} &= U_1 \times \dots \times U_k, \\ \mathbf{B} &= B_1 \times \dots \times B_k. \end{aligned}$$

For each state  $s \in M$ , every point  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  induces a set of models

$$\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})} = \left\{ M_1^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}, \dots, M_k^{(\mathbf{c}, \mathbf{u}, \mathbf{b})} \right\},$$

with  $M_i^{(\mathbf{c}, \mathbf{u}, \mathbf{b})} = \langle S_i, \alpha_i(s_0), \mathbf{u}[i], O_i, \varphi_i, \mathbf{c}[i], Q^{b[i]} \rangle :$

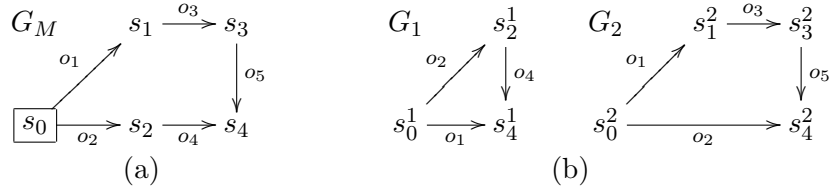


Figure 3: Illustration for our running example

- the states  $S_i$  and operators  $O_i$  correspond to the nodes and edge labels of  $G_i$ ;
- the transition function  $\varphi_i(s, o) = s'$  iff  $T_i$  contains an arc from  $s$  to  $s'$  labeled with  $o \in O_i$ ;
- the initial state  $\alpha_i(s_0)$  is determined by the initial state  $s_0$  and the state mapping  $\alpha_i$ ; and
- the operator cost function, state value function, and cost budget are all directly determined by the choice of  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$ .

For some choices  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$  from  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ , the induced sets of models  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  can be used for deriving admissible estimates for the state of interest  $s_0$ , while other cannot. The respective qualification is defined below.

**Definition 1 (Additive OSP Abstraction)**

Let  $M = \langle S, s_0, u, O, \varphi, c, Q^b \rangle$  be an OSP model, and  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  be an abstraction skeleton for  $M$ . For  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$ ,  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  is an **(additive) abstraction for  $M$** , denoted as

$$\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})} \in_{\mathcal{AS}} M,$$

if and only if

$$h^*(s_0, b) \leq h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0, b) \stackrel{\text{def}}{=} \sum_{i \in [k]} h_i^*(\alpha_i(s_0), \mathbf{b}[i]),$$

that is, when  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0, b)$  is an admissible estimate of  $h^*(s_0, b)$ .

In simple terms, a set of models forms an additive OSP abstraction if *jointly* it does not *underestimate* the *value* that can be obtained from the *initial state*, within the given cost budget. For example, let  $G_M$  in Figure 3a be the graphical skeleton of a state model  $M = \langle \{s_0, \dots, s_4\}, s_0, u, \{o_1, \dots, o_5\}, \varphi, c, Q^b \rangle$ , with  $c(o_i) = 1$  for all operators  $o_i$ ,  $b = 2$ , and  $u(s_i) = \mathbb{1}_{\{i=4\}}$ . Let  $\mathcal{AS} = \{(G_1, \alpha_1), (G_2, \alpha_2)\}$  be an abstraction skeleton for  $M$ , with  $G_1$  and  $G_2$  as in Figure 3b and with state mappings

$$\alpha_1(s_i) = \begin{cases} s_4^1, & i \in \{1, 3\} \\ s_i^1, & \text{otherwise} \end{cases},$$

$$\alpha_2(s_i) = \begin{cases} s_4^2, & i = 2 \\ s_i^2, & \text{otherwise} \end{cases}.$$



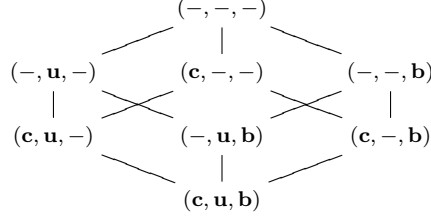


Figure 4: Fragments of restricted optimization over the abstractions  $\mathbf{A} \subseteq \mathbf{C} \times \mathbf{U} \times \mathbf{B}$

Consider a set of models  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$ , with constant  $\mathbf{c}[1](\cdot) = \mathbf{c}[2](\cdot) = 1$ ,  $\mathbf{b}[1] = \mathbf{b}[2] = 2$ , and, for  $j \in [2]$ ,  $\mathbf{u}[j](s_i^j) = \mathbb{1}_{\{i=5\}}$ . The optimal plan  $s_0$ -plan for  $M$  is  $\pi = \langle (s_0, o_2, s_2), (s_2, o_4, s_4) \rangle$ , with  $Q^b(\pi) = 1$ , while the optimal  $\alpha_1(s_0)$ -plan for  $M_1^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  is  $\pi_1 = \langle (s_0^1, o_1, s_4^1) \rangle$ , with  $Q^{\mathbf{b}[1]}(\pi_1) = 1$ , and the optimal  $\alpha_2(s_0)$ -plan for  $M_2^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  is  $\pi_2 = \langle (s_0^2, o_2, s_4^2) \rangle$ , with  $Q^{\mathbf{b}[2]}(\pi_2) = 1$ . Since

$$h^*(s_0, b) = Q^b(\pi) \leq Q^{\mathbf{b}[1]}(\pi_1) + Q^{\mathbf{b}[2]}(\pi_2) = h_1^*(\alpha_1(s_0), \mathbf{b}[1]) + h_2^*(\alpha_2(s_0), \mathbf{b}[2]),$$

$\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  is an additive abstraction for  $M$ .

**Theorem 1** *For any OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , any abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ , and any  $\mathcal{M} \in_{\mathcal{AS}} M_\Pi$ , if the digraphs of  $\mathcal{AS}$  are given explicitly, then  $h_{\mathcal{M}}(s_0, b)$  can be computed in time polynomial in  $\|\Pi\|$  and  $\|\mathcal{M}\|$ .*

*Proof:* The proof is straightforward. Let  $\mathcal{M} = \{M_i\}_{i \in [k]}$ , with  $M_i = \langle S_i, \alpha_i(s_0), u_i, O_i, \varphi_i, c_i, Q^{b_i} \rangle$ , be an additive abstraction for  $M_\Pi$  on the basis of  $\mathcal{AS}$ . For  $i \in [k]$ , let  $S'_i = \{s \in S_i \mid c_i(\alpha_i(s_0), s) \leq b_i\}$ . Since the digraphs of  $\mathcal{AS}$  are given explicitly, computing shortest paths from  $\alpha_i(s_0)$  to all states in  $G_i$ , and thus computing  $S'_i$ , can be done in time polynomial in  $\|\mathcal{M}\|$  for all  $i \in [k]$ . In turn, since  $h_i^*(\alpha_i(s_0), b_i) = \max_{s \in S'_i} u_i(s)$ , computing  $h_{\mathcal{M}}(s_0, b) = \sum_{i \in [k]} h_i^*(\alpha_i(s_0), b_i)$  is polynomial time in  $\|\mathcal{M}\|$ .  $\square$

The message of Theorem 1 is positive, yet it establishes only a necessary condition for the relevance of OSP abstractions to practice. Given an OSP task  $\Pi$ , and having fixed an abstraction skeleton  $\mathcal{AS}$  with a joint performance measure space  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ , we should be able to automatically separate between those  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  that constitute abstractions for  $M_\Pi$  and those that do not, and within the former set, denoted as

$$\mathbf{A} \subseteq \mathbf{C} \times \mathbf{U} \times \mathbf{B},$$

home in on an abstraction that provides us with as accurate (aka as *low*) an estimate of  $h^*(s_0, b)$  as possible. Here, even the first item on the agenda is not necessarily trivial as, in general,  $\mathbf{A}$  seem to lack convenient combinatorial properties. For instance, generally  $\mathbf{A}$  does not form a combinatorial rectangle in  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ : Consider the OSP state model  $G_M$  and abstraction skeleton  $\mathcal{AS}$  from our running example. Let  $\mathbf{c} \in \mathbf{C}$  be a cost function vector with both  $\mathbf{c}[1]$  and  $\mathbf{c}[2]$  being constant functions with value of 1, and two performance measures  $(\mathbf{c}, \mathbf{u}, \mathbf{b}), (\mathbf{c}, \mathbf{u}', \mathbf{b}') \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  being defined via budget vectors  $\mathbf{b} = \{\mathbf{b}[1] = 2, \mathbf{b}[2] = 0\}$

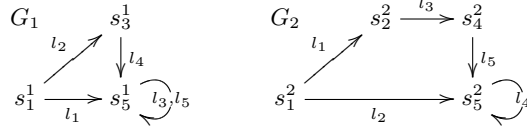


Figure 5: Homomorphic abstraction skeleton for  $G(\Pi)$  in Figure 3

and  $\mathbf{b}' = \{\mathbf{b}'[1] = 0, \mathbf{b}'[2] = 2\}$ , and value function vectors  $\mathbf{u}$  and  $\mathbf{u}'$ , with  $\mathbf{u}[1], \mathbf{u}[2], \mathbf{u}'[1]$ , and  $\mathbf{u}'[2]$  evaluating to zero on all states except for  $\mathbf{u}[1](s_5^1) = \mathbf{u}'[2](s_5^2) = 1$ . It is not hard to verify that  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b}), \mathcal{M}(\mathbf{c}, \mathbf{u}', \mathbf{b}') \in_{\mathcal{AS}} M$ , yet  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b}), \mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b}') \notin_{\mathcal{AS}} M$ .

Taking that on board, we break down and approach the overall agenda of complexity analysis of abstraction-based heuristic functions under *fixation* of some of the three dimensions of  $\mathbf{A}$ : If, for instance, we are *given* a vector of value functions  $\mathbf{u}$  that is *known* to belong to the projection of  $\mathbf{A}$  on  $\mathbf{U}$ , then we can search for a quality abstraction from the abstraction subset  $\mathbf{A}(-, \mathbf{u}, -) \subset \mathbf{A}$ , corresponding to the projection of  $\mathbf{A}$  on  $\{\mathbf{u}\}$ . As we show below, even some constrained optimizations of this kind can be challenging. The lattice in Figure 4 depicts the range of options for such constrained optimization; at the extreme settings,  $\mathbf{A}(-, -, -)$  is simply a renaming of  $\mathbf{A}$ , and  $\mathbf{A}(\mathbf{c}, \mathbf{u}, \mathbf{b})$  corresponds to a single abstraction  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}$ .

### 3.2 Partitions and Homomorphic Abstractions

We now proceed with considering a specific family of additive abstractions, reveal some of its interesting properties, and show that it contains substantial islands of tractability. With Definition 1 allowing for very general abstraction skeletons, in this work we focus on<sup>4</sup> *homomorphic abstraction skeletons* (Helmert et al., 2007).

**Definition 2** An abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  for an OSP state model  $M = \langle S, s_0, u, O, \varphi, c, Q^b \rangle$  is **homomorphic** if, for  $i \in [k]$ ,  $O_i = O$ , and  $\varphi(s, o) = s'$  only if  $(\alpha_i(s), o, \alpha_i(s')) \in T_i$ .

For instance, in our running example, the abstraction skeleton in Figure 3b is not homomorphic, while the abstraction skeleton in Figure 5 is homomorphic. Furthermore, we focus on a fragment of additive abstractions

$$\mathbf{A}_p = \mathbf{A} \cap [\mathbf{C}_p \times \mathbf{U}_p \times \mathbf{B}_p],$$

where  $\mathbf{C}_p \subseteq \mathbf{C}$ ,  $\mathbf{U}_p \subseteq \mathbf{U}$ , and  $\mathbf{B}_p \subseteq \mathbf{B}$  correspond to *cost*, *value*, and *budget partitions*, respectively.

**Definition 3** Given an OSP state model  $M = \langle S, s_0, u, O, \varphi, c, Q^b \rangle$ , and a homomorphic abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  for  $M$  with a joint performance measure  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ ,

4. All the results also hold verbatim for the more general “labeled paths preserving” abstraction skeletons studied by Katz and Domshlak (2010b) in the context of optimal classical planning. However, the presentation is somewhat more accessible when restricted to homomorphic abstraction skeletons.

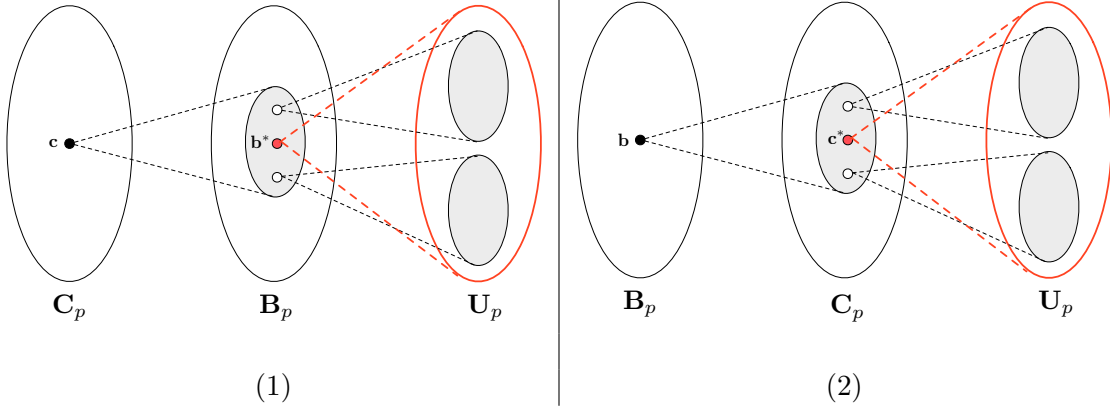


Figure 6: Illustration for sub-claims (1) and (2) of Theorem 2: In (1), the gray ellipse within  $\mathbf{B}_p$  stands for the subset of budget partitions  $\mathbf{b}$  that pair with  $\mathbf{c}$  in some abstraction, that is,  $\mathbf{A}_p(\mathbf{c}, -, \mathbf{b}) \neq \emptyset$ . However, while pairing some of these budget partitions  $\mathbf{b}$  with  $\mathbf{c}$  requires then a careful selection of a value partition  $\mathbf{u}$  (so that  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  will be an abstraction), there exists *some* budget partition  $\mathbf{b}^*$  for which any choice of  $\mathbf{u}$  will do the job.

- $\mathbf{c} \in \mathbf{C}$  is a **cost partition** iff, for each operator  $o \in O$ ,  $\sum_{i \in [k]} \mathbf{c}[i](o) \leq c(o)$ ;
- $\mathbf{u} \in \mathbf{U}$  is a **value partition** iff, for each state  $s \in S$ ,  $\sum_{i \in [k]} \mathbf{u}[i](\alpha_i(s)) \geq u(s)$ ; and
- $\mathbf{b} \in \mathbf{B}$  is a **budget partition** iff,  $\sum_{i \in [k]} \mathbf{b}[i] \leq b$ .

In what follows, for any node  $x$  of the lattice in Figure 4, by  $\mathbf{A}_p(x)$  we refer to  $\mathbf{A}(x) \cap \mathbf{A}_p$ ; e.g.,  $\mathbf{A}_p(-, \mathbf{u}, -) = \mathbf{A}(-, \mathbf{u}, -) \cap \mathbf{A}_p$ .

We begin our analysis of  $\mathbf{A}_p$  by establishing an interesting “completeness” relationship between the sets  $\mathbf{C}_p$  and  $\mathbf{B}_p$ , as well as an even stronger individual “completeness” of  $\mathbf{C}_p$  and  $\mathbf{B}_p$ . Formulated in Theorem 2, these properties of  $\mathbf{A}_p$  play a key role in our computational analysis later on.

**Theorem 2** *Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$  and a homomorphic abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ ,*

- (1) *for each cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b}^* \in \mathbf{B}_p$  such that  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)} \in_s \mathcal{AS}$  for all value partitions  $\mathbf{u} \in \mathbf{U}_p$ .*
- (2) *for each budget partition  $\mathbf{b} \in \mathbf{B}_p$ , there exists a cost partition  $\mathbf{c}^* \in \mathbf{C}_p$  such that  $\mathcal{M}^{(\mathbf{c}^*, \mathbf{u}, \mathbf{b})} \in_s \mathcal{AS}$  for all value partitions  $\mathbf{u} \in \mathbf{U}_p$ .*

The proof of Theorem 2 appears in Appendix A, p. 44. Figure 6 illustrates the statement of sub-claim (1) of Theorem 2, as well as, indirectly, some of its corollaries.<sup>5</sup> The first corollary of Theorem 2 is that the projections of  $\mathbf{A}_p$  on  $\mathbf{C}_p$ ,  $\mathbf{U}_p$ , and  $\mathbf{B}_p$  are the entire sets  $\mathbf{C}_p$ ,

5. The respective illustration of sub-claim (2) of Theorem 2 is completely similar, *mutatis mutandis*.

$\mathbf{U}_p$ , and  $\mathbf{B}_p$ , respectively. That is, any cost partition  $\mathbf{c}$  (and similarly, any budget partition and any value partition) can be matched with an abstraction that has that partition as its component. Second, while not any budget partition  $\mathbf{b}$  can be paired with a given cost partition  $\mathbf{c}$  in abstractions for  $M_\Pi$ , that is, not for all  $\mathbf{b} \in \mathbf{B}_p$ ,  $\mathbf{A}_p(\mathbf{c}, -, \mathbf{b}) \neq \emptyset$ , there are always *some* budget partitions that can be paired with  $\mathbf{c}$ . Finally, while pairing some of these “ $\mathbf{c}$ -compatible” budget partitions  $\mathbf{b}$  with  $\mathbf{c}$  requires then a careful selection of a value partition  $\mathbf{u}$ , there exists *some* “ $\mathbf{c}$ -compatible” budget partition  $\mathbf{b}^*$  for which any choice of  $\mathbf{u}$  will result in  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})$  being an abstraction of  $M_\Pi$ .

A priori, these properties of  $\mathbf{A}_p$  should simplify the task of abstraction discovery and optimization within the space of partitions  $\mathbf{C}_p \times \mathbf{U}_p \times \mathbf{B}_p$ , and later we show that this is indeed the case. However, complexity analysis of abstraction discovery within  $\mathbf{C}_p \times \mathbf{U}_p \times \mathbf{B}_p$  in most general terms is still problematic because OSP formalism is parametric in the representation of value functions. Hence, here we proceed with examining abstraction discovery for OSP in the context of *fixed* value partitions  $\mathbf{u} \in \mathbf{U}_p$ .

## 4. From Value Partitions to Complete Abstractions

Let  $\Pi$  be an OSP task,  $\mathcal{AS}$  be an explicitly given homomorphic abstraction skeleton of  $M_\Pi$ , and  $\mathbf{u} \in \mathbf{U}_p$  be a value partition over  $\mathcal{AS}$ . An immediate corollary of Theorem 2 is that  $\mathbf{A}_p(-, \mathbf{u}, -)$  is not empty, and thus we can try computing  $\min_{(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(-, \mathbf{u}, -)} h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0)$ . As of yet, however, we do not know whether this task is polynomial-time solvable for any non-trivial class of value partitions. In fact, despite that, by Theorem 2,  $\mathbf{A}_p(-, \mathbf{u}, -)$  is known to be non-empty, and so, too, are all of its subsets  $\mathbf{A}_p(-, \mathbf{u}, \mathbf{b})$  and  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$ , finding even just *any* abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(-, \mathbf{u}, -)$  is not necessarily easy.

### 4.1 0-Binary Value Partitions

As a first step, we now examine abstraction discovery within a fragment of  $\mathbf{A}_p$  in which all value functions  $\mathbf{u}[i]$  of the abstract models are what we call *0-binary*. Later, in Section 4.2, we show how our findings for 0-binary abstract value functions can be extended to general value partitions.

**Definition 4** *A real-valued function  $f$  is called **0-binary** if it is to  $\{0, \sigma\}$  for some  $\sigma \in \mathbb{R}^+$ . A set  $F$  of 0-binary functions is called **strong** if all the functions in  $F$  are to  $\{0, \sigma\}$  for the same  $\sigma \in \mathbb{R}^+$ .*

On the one hand, 0-binary functions constitute rather a basic family of value functions. Hence, if abstraction optimization is hard for them, it is likely to be hard for any non-trivial family of abstract value functions. On the other hand, 0-binary abstract value functions seem to fit well abstractions of planning tasks in which value functions are linear combinations of indicators, each representing achievement of a “goal value” for some state variable.

In that respect, our first tractability results are for abstraction discovery within  $\mathbf{A}_p(-, \mathbf{u}, -)$  where  $\mathbf{u}$  is a *strong* 0-binary value partition. The first (and the simpler) result in Theorem 3 further assumes a fixed action cost partition, while the next result, in Theorem 7, is on simultaneous selection of admissible pairs of cost and budget partitions. In Corollary 4

and Theorem 10 we then show how the results of Theorem 3 and Theorem 7, respectively, can be extended to pseudo-polynomial algorithms for *general* 0-binary value partitions.

#### 4.1.1 STRONG 0-BINARY VALUE PARTITIONS AND THE KNAPSACK PROBLEM

Our first tractability result is for abstraction discovery within  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  where  $\mathbf{u}$  is a *strong* 0-binary value partition and  $\mathbf{c}$  is an arbitrary cost partition. The key role here is played by the well-known Knapsack problem (Dantzig, 1930; Kellerer, Pferschy, & Pisinger, 2004). An instance  $\langle \{w_i, \sigma_i\}_{i \in [n]}, W \rangle$  of the **Knapsack** problem is given by a weight allowance  $W$  and a set of objects  $[n]$ , with each object  $i \in [n]$  being annotated with a weight  $w_i$  and a value  $\sigma_i$ . The objective is to find a subset  $Z \subseteq [n]$  that maximizes  $\sum_{i \in Z} \sigma_i$  over all subsets  $Z' \subseteq [n]$  with  $\sum_{i \in Z'} w_i \leq W$ . By **strict Knapsack** we refer to a variant of Knapsack in which that inequality constraint is strict. Knapsack is NP-hard (Karp, 1972; Garey & Johnson, 1978), but there exist pseudo-polynomial algorithms for it that run in time polynomial in the description of the problem and in the unary representation of  $W$  (Dudzinski & Walukiewicz, 1987). The latter property makes solving Knapsack practical in many applications where the ratio  $\frac{W}{\min_i w_i}$  is reasonably low. Likewise, if  $\sigma_i = \sigma_j$  for all  $i, j \in [n]$ , then a greedy algorithm solves the problem in linear time by iteratively expanding  $Z$  by one of the weight-wise lightest objects in  $[n] \setminus Z$ , until  $Z$  cannot be expanded any further within  $W$ .

#### Theorem 3 ( $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$ & strong 0-binary $\mathbf{u}$ )

Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{AS}$  be an explicit homomorphic abstraction skeleton of  $M_\Pi$ , and  $\mathbf{u} \in \mathbf{U}_p$  be a strong 0-binary value partition. Given a cost partition  $\mathbf{c} \in \mathbf{C}_p$ , finding an abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  and computing the corresponding heuristic estimate  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0, b)$  can be done in time polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ .

*Proof:* The proof is by reduction to the polynomial fragment of the Knapsack problem corresponding to all items having identical value. Let  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$ , and, given that  $\mathbf{u}$  is a strong 0-binary value partition, let all  $\mathbf{u}[i]$  be to  $\{0, \sigma\}$  for some  $\sigma \in \mathbb{R}^+$ .

For  $i \in [k]$ , let  $w_i$  be the cost of the cheapest path in  $G_i$  from  $\alpha_i(s_0)$  to (one of the) states  $s \in S_i$  with  $\mathbf{u}[i](s) = \sigma$ . Since  $\mathcal{AS}$  is an *explicit* abstraction skeleton, the set  $\{w_i\}_{i \in [k]}$  can be computed in time polynomial in  $\|\mathcal{AS}\|$  using one of the standard algorithms for the single-source shortest paths problem. Consider now a Knapsack problem  $\langle \{w_i, \sigma\}_{i \in [k]}, b \rangle$ , with weights  $w_i$  being as above and value  $\sigma$  being identical for all objects. Let  $Z \subseteq [k]$  be a solution to that (optimization) Knapsack problem; recall that it is computable in polynomial time. Given that, we define budget profile  $\mathbf{b}^* \in \mathbf{B}$  as follows:

$$\text{for } i \in [k], \mathbf{b}^*[i] = \begin{cases} w_i, & i \in Z \\ 0, & \text{otherwise.} \end{cases}$$

What remains to be shown is that  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  actually induces an additive abstraction for  $M_\Pi$ . Assume to the contrary that  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)} \notin_{\mathcal{AS}} M_\Pi$ , and let  $\pi$  be an optimal  $s_0$ -plan for  $\Pi$ . By the construction of our Knapsack problem and of  $\mathbf{b}^*$ , for each  $i \in Z$ , there is a  $\alpha_i(s)$ -plan  $\pi_i$  for  $M_i^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)}$  with  $Q^{\mathbf{b}^*[i]}(\pi_i) = \sigma$ . By Definition 1, our assumption implies that  $Q^b(\pi) > \sum_{i \in Z} Q^{\mathbf{b}^*[i]}(\pi_i) = \sigma \cdot |Z|$ . However, by Theorem 2, there exists at least one budget partition  $\mathbf{b} \in \mathbf{B}_p$  such that  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})} \in_{\mathcal{AS}} M_\Pi$ . Note that this budget partition

induces a feasible solution  $Z' = \{i \mid w_i \leq \mathbf{b}[i]\}$  for our Knapsack problem, satisfying  $Q^b(\pi) \leq \sum_{i \in Z'} Q^{\mathbf{b}[i]}(\pi_i) = \sigma \cdot |Z'|$ . This, however, implies  $|Z| < |Z'|$ , contradicting the optimality of  $Z$ , and thus accomplishing the proof that  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)} \in_{\mathcal{AS}} M_\Pi$ .  $\square$

The construction in the proof of Theorem 3 may appear somewhat counterintuitive: while we are interested in minimizing the heuristic estimate of  $h^*(s_0, b)$ , the abstraction  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)}$  is selected via the value-maximizing Knapsack problem. Indeed, while ultimately we would like to obtain

$$\min_{\mathbf{b} : (\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p} h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0, b), \quad (6)$$

the heuristic we manage to compute in polynomial time is actually

$$\max_{\mathbf{b} : (\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p} h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0, b). \quad (7)$$

As it can be seen from the proof of Theorem 3, Eq. 7 seems to be the best one can hope for to achieve on the basis of  $\mathbf{A}_p$ 's properties captured by Theorem 2. However, note that, for a fixed pair of  $\mathbf{c} \in \mathbf{C}_p$  and  $\mathbf{u} \in \mathbf{U}_p$ , this estimate in Eq. 7 is still at least as (and possibly much more) accurate as the estimate that would be obtained by providing each of the  $k$  abstract models with the entire budget  $b$ . Shortly below we show that this superior accuracy clearly shows up in our experiments, but first we proceed with examining working with general 0-binary value partitions.

While strong 0-binary value partitions are rather restrictive, finding an element of  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  for general 0-binary  $\mathbf{u}$  is no longer polynomial—a reduction from Knapsack is straightforward. However, Knapsack is solvable in pseudo-polynomial time, and plugging that Knapsack algorithm into the proof of Theorem 3 results in a search algorithm for  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  with general 0-binary  $\mathbf{u}$ .

#### Corollary 4 ( $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$ & 0-binary $\mathbf{u}$ )

*Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{AS}$  be an explicit homomorphic abstraction skeleton of  $M_\Pi$ , and  $\mathbf{u} \in \mathbf{U}_p$  be a 0-binary value partition. Given a cost partition  $\mathbf{c} \in \mathbf{C}_p$ , finding an abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  and computing the corresponding heuristic estimate  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0, b)$  can be done in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ , and the unary representation of the budget  $b$  of  $\Pi$ .*

To test and illustrate the value that additive abstractions can bring to heuristic-search OSP, we have implemented a prototype heuristic-search OSP solver on top of the Fast Downward planner (Helmert, 2006).<sup>6</sup> Since, unlike classical and net-benefit planning, OSP still lacks a standard suite of benchmarks for comparative evaluation, we have cast in this role the STRIPS classical planning domains from the International Planning Competitions (IPC) 1998-2006. This “translation” to OSP was done by associating a separate unit-value with each sub-goal.

Within our prototype, we have implemented the *BFB* search for OSP, and provided support for some basic pattern-database abstraction skeletons, action cost partitions, and abstraction selection in  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  for strong 0-binary value partitions as in the proof of Theorem 3. Specifically, for a task with  $k$  sub-goals,

6. We are not aware of any other domain-independent planner for optimal OSP.

	25%			50%			75%			100%		
	$h_{\mathcal{M}}$	<i>basic</i>	<i>blind</i>	$h_{\mathcal{M}}$	<i>basic</i>	<i>blind</i>	$h_{\mathcal{M}}$	<i>basic</i>	<i>blind</i>	$h_{\mathcal{M}}$	<i>basic</i>	<i>blind</i>
airport (25)	23	23	23	20	20	20	19	20	18	19	20	18
blocks (23)	23	23	23	23	23	23	22	18	17	17	17	17
depot (3)	3	3	3	3	3	3	3	3	3	3	2	2
driverlog (12)	12	12	12	12	12	11	11	9	9	10	7	6
freecell (5)	5	5	5	5	5	5	5	5	5	5	5	5
grid (2)	2	2	2	2	2	2	2	2	2	1	1	1
gripper (6)	6	6	6	6	6	6	6	6	6	6	6	6
logistics (10)	10	10	10	10	10	10	10	10	10	10	10	10
miconic (50)	50	50	50	50	50	50	50	50	50	50	45	45
mystery (4)	4	4	4	4	4	4	4	4	4	3	3	2
openstacks (7)	7	7	7	7	7	7	7	7	7	7	7	7
rovers (10)	10	10	10	10	7	7	7	6	6	6	5	5
satellite (9)	9	8	8	7	6	6	6	4	5	5	4	4
tpp (7)	7	7	7	7	7	7	6	6	6	6	5	5
trucks (9)	9	9	9	9	8	8	6	5	5	5	5	5
pipesw-t (12)	12	12	12	12	12	12	12	11	11	11	10	10
pipesw-nt (7)	7	7	7	7	7	7	7	7	7	7	6	6
psr-small (30)	30	30	30	30	30	30	30	30	30	30	30	30
zenotravel (10)	10	10	10	10	9	8	9	8	8	8	7	7
total	239	238	238	234	228	226	222	211	209	209	195	191

Table 1: Number of problems solved across the different budgets using the OPEN list ordered by the heuristic evaluation as in Figure 2

- (i) the abstraction skeleton comprised a set of some  $k$  projections of the planning task onto connected subsets of ancestors of the respective  $k$  goal variables in the causal graph;
- (ii) the value partition  $\mathbf{u}$  associated the value of each sub-goal (only) with the respective projection; and
- (iii) an ad hoc action cost partition  $\mathbf{c}$ .

The size of each projection was limited to 1000 abstract states.

In our evaluation, we compared *BFBB* node expansions with three heuristic functions, tagged *blind*, *basic*, and  $h_{\mathcal{M}}$ . With all three heuristics, the  $h$ -value of a node  $n$  is set to 0 if the cost budget at  $n$  is over-consumed. Otherwise,

- *blind BFBB* constitutes a trivial baseline in which  $h(n)$  is simply set to the total value of all goals.
- In *basic BFBB*,  $h(n)$  is set to the total value of goals, each of which can be *individually* achieved within the respective projection abstraction (see Theorem 1) given the *entire remaining budget*.
- $h_{\mathcal{M}}$  is an additive abstraction heuristic that is selected from  $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  as in the proof of Theorem 3.

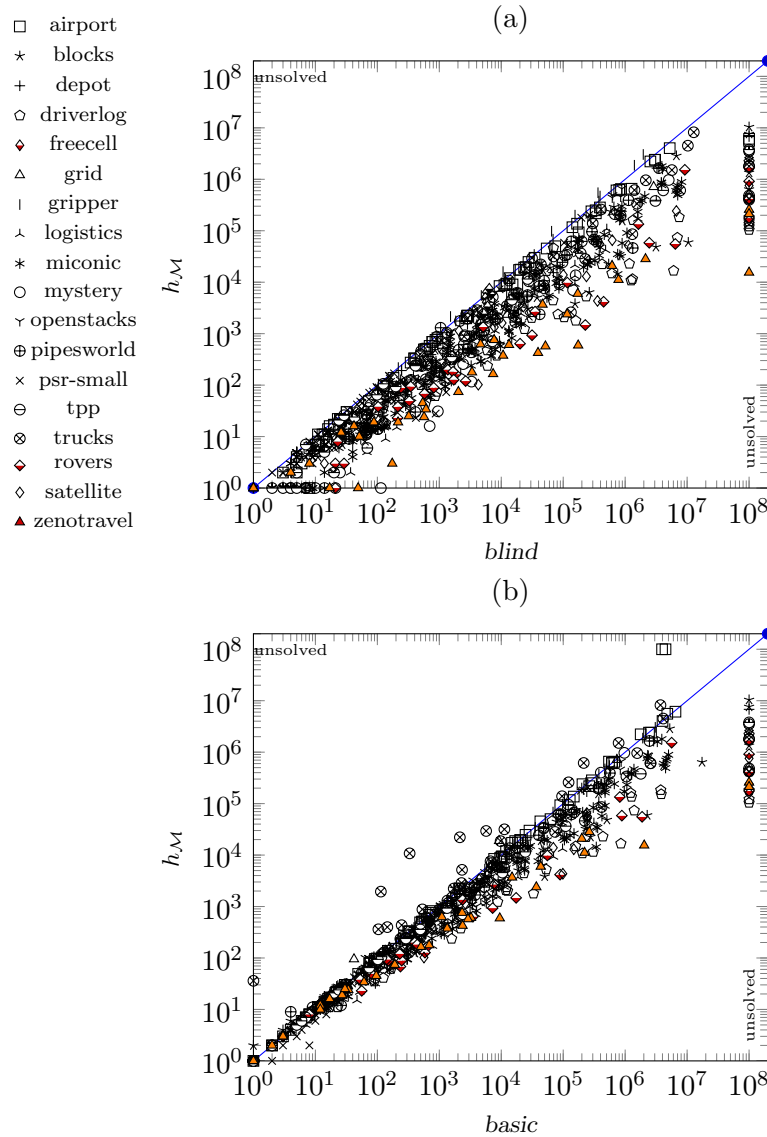


Figure 7: Comparative view of empirical results from Table 1 in terms of expanded nodes

The evaluation contained all the planning tasks for which we could determine offline the minimal cost needed to achieve all the goals. Each such task was approached under four different budgets, corresponding to 25%, 50%, 75%, and 100% of the minimal cost needed to achieve all the goals in the task, and each run was restricted to 10 minutes. Table 1 shows the number of tasks solved within each domain for each level of cost budget, and Figure 7 depicts the results in terms of expanded nodes across the four levels of cost budget. (Figures 15-18 in Appendix B provide a more detailed view on the results in Figure 7 by breaking them along different levels of cost budget.) Despite the simplicity of the abstraction skeletons we used, the number of nodes expanded by *BFBB* with  $h_M$  was



typically substantially lower than the number of nodes expanded by *basic BFBB*, with the difference sometimes reaching three orders of magnitude.

#### 4.1.2 FREEING COST PARTITION: KNAPSACK MEETS CONVEX OPTIMIZATION

Returning now to the algorithmic analysis in the context of strong 0-binary value partitions, we now proceed with relaxing the constraint of sticking to a fixed action cost partition  $\mathbf{c}$ . This buys more flexibility in selecting abstractions from  $\mathbf{A}_p(-, \mathbf{u}, -)$  allowing to improve the accuracy of the heuristic estimates, while still remaining computationally tractable.

Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , a homomorphic abstraction skeleton  $\mathcal{AS}$ , and a value partition  $\mathbf{u} \in \mathbf{U}_p$  over  $\mathcal{AS}$ , let

$$\kappa(\mathbf{u}) = \min_{\mathbf{c} \in \mathbf{C}_p} \left[ \max_{\mathbf{b} : (\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p} h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0, b) \right]. \quad (8)$$

Obviously, the estimate  $h(s_0, b) = \kappa(\mathbf{u})$  is at least as accurate as the estimate in Eq. 7 that is derived with respect to a fixed cost partition  $\mathbf{c}$ .

We now show that, for any OSP task  $\Pi$ , any abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$  and any strong 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$  over  $\mathcal{AS}$ , computing  $\kappa(\mathbf{u})$  is polynomial time. The corresponding algorithm is shown in Figure 8, with Figure 8a depicting the macro-flow of the algorithm and Figure 8b depicting the specific implementation of the *solve* sub-routine that makes the overall time complexity of the algorithm polynomial.

The high-level flow of the algorithm in Figure 8a is as follows. Since  $\mathbf{u}$  is a strong 0-binary value partition, let all abstract value functions  $\mathbf{u}[i]$  be to  $\{0, \sigma\}$  for some  $\sigma \in \mathbb{R}^+$ . Given that, for each  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(-, \mathbf{u}, -)$ , it holds that  $h_{(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s) = m\sigma$  for some  $m \in \{0\} \cup [k]$ . The first, preprocessing for-loop of the algorithm eliminates from the abstraction skeleton all the nodes that are structurally unreachable from the abstract initial states  $\alpha_1(s_0), \dots, \alpha_k(s_0)$ .<sup>7</sup> For ease of presentation, in what follows we assume that this clean up of the abstraction skeleton leaves each  $G_i$  with at least one state whose value is  $\sigma$ . The second, main for-loop of the algorithm decreasingly iterates over all the values  $\{k\sigma, (k-1)\sigma, \dots, 2\sigma, \sigma\}$  that can possibly come from the abstractions in  $\mathbf{A}_p(-, \mathbf{u}, -)$  as an estimate of  $h^*(s_0, b)$ . Each of these candidates for  $\kappa(\mathbf{u})$  is tested in turn via the sub-routine *always-achievable*. If and when this test comes positive for the first time, then we are done, and the tested candidate  $m\sigma$  is identified as  $\kappa(\mathbf{u})$ . Otherwise, if the test fails for all  $m \in [k]$ , then  $\kappa(\mathbf{u}) = 0$ , in particular implying that no state with value greater than 0 can be reached from  $s_0$  under budget  $b$ .

The test of *always-achievable* for  $\kappa(\mathbf{u}) = m\sigma$  is based on a linear program (LP)  $\mathcal{L}_1(m)$ , given by Eq. 10. This linear program is defined over variables

$$\mathcal{X} = \{\xi\} \cup \bigcup_{i \in [k]} \left[ \{d(s)\}_{s \in G_i} \cup \{b[i]\} \cup \bigcup_{o \in O} \{c[i](o)\} \right], \quad (9)$$

constraints (10a)-(10c), and the objective of maximizing the value of the variable  $\xi$ .

7. This preprocessing can be replaced by adding some extra constraints in the linear program described below. However, that would unnecessary complicate the presentation without adding much value.

input:  $\Pi = \langle V, s_0, u; O, c, b \rangle$ ,  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ ,  
strong 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$   
output:  $\kappa(\mathbf{u})$

**for**  $i = 1$  **to**  $k$  **do**  
    reduce  $G_i$  to only nodes reachable from  $\alpha_i(s_0)$   
**for**  $m = k$  **downto** 1 **do**  
    **if** always-achievable( $m$ ) **then return**  $m\sigma$   
**return** 0

always-achievable( $m$ ):  
    solve( $\mathcal{L}_1(m)$ )  $\mapsto$  solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$   
    **if**  $\mathbf{x}[\xi] \leq b$  **then return** true  
    **else return** false  
(a)

solve( $\mathcal{L}_1(m)$ ):  
    set (10c') to an arbitrary subset of constraints (10c)  
    **loop**  
        set  $\mathcal{L}'_1(m)$  to  $\mathcal{L}_1(m)$ , with constraints (10c') instead of (10c)  
        ellipsoid-method( $\mathcal{L}'_1(m)$ )  $\mapsto$  solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$   
        let  $\tau$  be a permutation of  $[k]$  such that  
             $\mathbf{x}[\mathbb{b}[\tau(1)]] \leq \mathbf{x}[\mathbb{b}[\tau(2)]] \leq \dots \leq \mathbf{x}[\mathbb{b}[\tau(k)]]$   
        **if**  $\mathbf{x}[\xi] \leq \sum_{i \in [m]} \mathbf{x}[\mathbb{b}[\tau(i)]]$  **then return**  $\mathbf{x}$   
        extend (10c') with constraint  $\xi \leq \sum_{i \in [m]} \mathbb{b}[\tau(i)]$   
(b)

Figure 8: A polynomial-time algorithm for computing  $\kappa(\mathbf{u})$  for a strong 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$  (Theorem 7).

$\mathcal{L}_1(m)$  :

    max  $\xi$   
    subject to

$$\forall i \in [k] : \begin{cases} d(\alpha_i(s)) = 0, \\ d(s) \leq d(s') + \mathbb{c}[i](o), \quad \forall (s', a, s) \in G_i \\ \mathbb{b}[i] \leq d(s), \quad \forall s \in G_i \text{ s.t. } \mathbf{u}[i](s) = \sigma \end{cases}, \quad (10a)$$

$$\forall o \in O : \begin{cases} \mathbb{c}[i](o) \geq 0, \\ \sum_{i \in [k]} \mathbb{c}[i](o) \leq c(o) \end{cases} \quad \forall i \in [k], \quad (10b)$$

$$\forall Z \subseteq [k], |Z| = m : \xi \leq \sum_{i \in Z} \mathbb{b}[i], \quad (10c)$$

The roles of the different variables in  $\mathcal{L}_1(m)$  are as follows.

- Variable  $\mathbf{c}[i](o)$  captures the cost to be associated with label  $o$  in the digraph  $G_i$  of  $\mathcal{AS}$ .
- For a state  $s$  in  $G_i$ , variable  $d(s)$  captures the cost of the cheapest path in  $G_i$  from  $\alpha_i(s_0)$  to  $s$ , *given* that the edges of  $G_i$  are weighted consistently with the values of the variables  $\mathbf{c}[i](\cdot)$ .
- Variable  $\mathbf{b}[i]$  captures the minimal budget needed for reaching in  $G_i$  a state with value  $\sigma$  from state  $\alpha_i(s_0)$ , *given* that, again, the edges of  $G_i$  are weighted consistently with the variable vector  $\mathbf{c}[i]$ .
- The singleton variable  $\xi$  captures the minimal total cost of reaching states with value  $\sigma$  in *precisely*  $m$  out of  $k$  models in  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$ .

The semantics of the constraints in  $\mathcal{L}_1(m)$  are as follows.

- The first two sets of constraints in (10a) come from a simple LP formulation of the single source shortest paths problem with the source node  $\alpha_i(s_0)$ : Optimizing  $\sum_{i \in [k]} \sum_{s \in G_i} d(s)$  under a fixed weighting  $\mathbf{c}$  of the edges leads to computing precisely that, for all  $k$  digraphs in  $\mathcal{AS}$  simultaneously.
- The third set of constraints in (10a) establishes the costs of the cheapest paths in  $\{G_i\}$  from states  $\alpha_i(s_0)$  to states valued  $\sigma$ , enforcing the semantics of variables  $\mathbf{b}[1], \dots, \mathbf{b}[k]$ .
- Constraints (10b) are the cost partition constraints that enforce  $\mathbf{c} \in \mathbf{C}_p$ .
- Constraints (10c) enforce the aforementioned semantics of the objective variable  $\xi$ .

Two things are worth noting here. First, if all the nodes in the digraphs  $G_1, \dots, G_k$  are structurally reachable from the “source nodes”  $\alpha_1(s_0), \dots, \alpha_k(s_0)$ , respectively (as it is ensured by the first for-loop of the algorithm), then the polytope induced by  $\mathcal{L}_1(m)$  is bounded and non-empty. Indeed, for any assignment to  $\bigcup_{o \in O} \{\mathbf{c}[i](o)\}$  that is consistent with the positiveness constraints in (10b), all the variables  $d(\cdot)$  are bounded from above by the lengths of the respective shortest paths. In turn, this bounding of  $d(\cdot)$  bounds from above the variables  $\mathbf{c}[1], \dots, \mathbf{c}[k]$  via the third set of constraints in (10a), and the constraints (10c) then bound from above the objective  $\xi$ .

Second, while the number of variables, as well as the number of constraints in (10a) and (10b), are polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ , the number of constraints in (10c) is  $\binom{k}{m}$ . Thus, solving  $\mathcal{L}_1(m)$  using standard methods for linear programming is not practical. In Lemma 5 below we show that this issue can actually be mitigated, but then, in Lemma 6 we show that the semantics of  $\mathcal{L}_1(m)$  match our objective of finding  $\kappa(\mathbf{u})$ .

**Lemma 5** *The algorithm in Figure 8 terminates in time polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ .*

*Proof:* The runtime complexity of the algorithm boils down to the complexity of solving  $\mathcal{L}_1(m)$ , and, while the number of variables in  $\mathcal{L}_1(m)$ , as well as the number of constraints in (10a) and (10b), are polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ , the number of constraints in (10c) is

$\binom{k}{m}$ ). Thus, solving  $\mathcal{L}_1(m)$  using standard methods for linear programming is not practical. However, using the ellipsoid algorithm for linear inequalities (Grotschel, Lovasz, & Schrijver, 1981), an LP with an exponential number of constraints can be solved in polynomial time provided that an associated “separation problem” can be solved in polynomial time. In our case, the separation problem is, given an assignment to the variables of  $\mathcal{L}_1(m)$ , test whether it satisfied (10a), (10b), and (10c), and if not, produce an inequality among (10a), (10b), and (10c) violated by that assignment.

We now show how our separation problem for  $\mathcal{L}_1(m)$  can be solved in polynomial time using what is called  $m$ -sum minimization LPs (Punnen, 1992), and this is precisely what the procedure `solve`( $\mathcal{L}_1(m)$ ) in Figure 8b does. As the number of constraints in (10a) and (10b) is polynomial, their satisfaction by an assignment  $\mathbf{x} \in \text{dom}(\mathcal{X})$  can be tested directly by substitution. For constraints (10c), let  $\tau$  be a permutation of  $[k]$  such that  $\mathbf{x}[\mathbf{b}[\tau(1)]] \leq \mathbf{x}[\mathbf{b}[\tau(2)]] \leq \dots \leq \mathbf{x}[\mathbf{b}[\tau(k)]]$ . If  $\mathbf{x}[\xi] \leq \sum_{i \in [m]} \mathbf{x}[\mathbf{b}[\tau(i)]]$ , then it is easy to see that  $\mathbf{x}$  satisfies all the constraints in (10c). Otherwise, we have our violated inequality  $\xi \leq \sum_{i \in [m]} \mathbf{b}[\tau(i)]$ .  $\square$

**Lemma 6** *The algorithm in Figure 8a computes  $\kappa(\mathbf{u})$ .*

The proof of Lemma 6 appears in Appendix A, p. 45. Putting Lemmas 5 and 6 together, Theorem 7 summarizes our tractability result for abstraction discovery in  $\mathbf{A}_p(-, \mathbf{u}, -)$  for strong 0-binary value partitions  $\mathbf{u}$ .

**Theorem 7** ( $\mathbf{A}_p(-, \mathbf{u}, -)(s)$  & strong 0-binary  $\mathbf{u}$ )

*Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS}$  of  $M_\Pi$ , and a strong 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$ , computing  $\kappa(\mathbf{u})$  can be done in time polynomial in  $||\Pi||$  and  $||\mathcal{AS}||$ .*

#### 4.1.3 FROM STRONG TO GENERAL 0-BINARY VALUE PARTITIONS

Recall that the polynomial result of Theorem 3 for strong 0-binary value partitions easily extends in Corollary 4 to a pseudo-polynomial algorithm for general 0-binary value partitions. It turns out that a pseudo-polynomial extension of Theorem 7 is possible as well, though it is technically more involved. The corresponding algorithm is shown in Figure 9. Following the format of Figure 8, Figure 9a depicts the macro-flow of the algorithm and Figure 9b shows the specific implementation of the `solve` sub-routine that allows achieving the desired time complexity.

Similarly to the algorithm in Figure 8, first, a preprocessing for-loop of the algorithm eliminates from the abstraction skeleton all the nodes that are structurally unreachable from the abstract initial states  $\alpha_1(s_0), \dots, \alpha_k(s_0)$ . Next, the algorithm performs a binary search over an interval containing  $\kappa(\mathbf{u})$ .<sup>8</sup> Since  $\mathbf{u}$  is a 0-binary value partition, for  $i \in [k]$ , by  $\{0, \sigma_i\}$ ,  $\sigma_i \in \mathbb{R}^+$ , we denote the range of the abstract value function  $\mathbf{u}[i]$ . Given that, for each  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(-, \mathbf{u}, -)$ , it holds that  $h_{(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s) = \sum_{i \in Z} \sigma_i$  for some  $Z \subseteq [k]$ . As the size of this combinatorial hypothesis space is prohibitive, the while-loop in Figure 9

8. While a binary search could have been used in the algorithm in Figure 8 as well, there it would be a mere optimization, while here it is necessary to avoid an exponential blowup of the time complexity.

input:  $\Pi = \langle V, s_0, u; O, c, b \rangle$ ,  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ ,  
0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$   
output:  $\kappa_s(\mathbf{u})$

**for**  $i = 1$  **to**  $k$  **do**  
    reduce  $G_i$  to only nodes reachable from  $\alpha_i(s_0)$   
**let**  $0 < \epsilon < \min_{i \in [k]} \sigma_i$   
 $\alpha \leftarrow 0$   
 $\beta \leftarrow \sum_{i \in [k]} \sigma_i$   
**while**  $\beta - \alpha > \epsilon$  **do**  
     $v \leftarrow \alpha + (\beta - \alpha)/2$   
    **if**  $\text{always-achievable}(v)$  **then**  $\alpha \leftarrow v$   
    **else**  $\beta \leftarrow v$   
**if**  $\alpha = 0$  **then return** 0  
**else return**  $\beta$

$\text{always-achievable}(v)$ :  
     $\text{solve}(\mathcal{L}_2(v)) \mapsto \text{solution } \mathbf{x} \in \text{dom}(\mathcal{X})$   
    **if**  $\mathbf{x}[\xi] \leq b$  **then return** true  
    **else return** false  
(a)

$\text{solve}(\mathcal{L}_2(v))$ :  
    set (11c') to an arbitrary subset of constraints (11c)  
    **loop**  
        set  $\mathcal{L}'_2(v)$  to  $\mathcal{L}_2(v)$ , with constraints (11c') instead of (11c)  
         $\text{ellipsoid-method}(\mathcal{L}'_2(v)) \mapsto \text{solution } \mathbf{x} \in \text{dom}(\mathcal{X})$   
         $\text{strict-Knapsack}(\langle \{\mathbf{x}[\mathbb{b}[i]], \sigma_i\}_{i \in [k]}, \mathbf{x}[\xi] \rangle) \mapsto \text{solution } Z \subseteq [k]$   
        **if**  $\sum_{i \in Z} \sigma_i < v$  **then return**  $\mathbf{x}$   
        extend (11c') with constraint  $\xi \leq \sum_{i \in Z} \mathbb{b}[i]$   
    (b)

Figure 9: A pseudo-polynomial algorithm for approximating  $\kappa(\mathbf{u})$  for general 0-binary value partitions  $\mathbf{u} \in \mathbf{U}_p$  (Theorem 10).

performs a binary search over a relaxed hypothesis space, corresponding to the continuous interval  $[0, \sum_{i \in [k]} \sigma_i]$  of  $\mathbb{R}^{+0}$ . The parameter  $\epsilon$  serves as the “sufficient precision” criterion for termination.

At iteration corresponding to an interval  $[\alpha, \beta]$ , the algorithm uses its sub-routine  $\text{always-achievable}$  to test the hypothesis  $\kappa(\mathbf{u}) \geq v$ , where  $v$  is the mid-point of  $[\alpha, \beta]$ . If the test comes positive, then the next tested hypothesis is  $\kappa_s(\mathbf{u}) \geq v'$ , where  $v'$  is the mid-point of  $[v, \beta]$ . Otherwise, the next hypothesis corresponds to the midpoint of  $[\alpha, v]$ . When the while-loop is done, the reported estimate is set to  $\beta$ ; while there still might be some lag between  $\beta$  and  $\kappa(\mathbf{u})$ , this lag can be arbitrarily reduced by reducing  $\epsilon$ , and anyway,

$\beta \geq \kappa(\mathbf{u})$  ensures admissibility of the estimate. If, however, the while-loop terminates with  $\alpha = 0$ , then  $\kappa(\mathbf{u}) \leq \beta \leq \epsilon < \min_{i \in [k]} \sigma_i$  implies  $\kappa(\mathbf{u}) = 0$ , and this is what we return.

The test of **always-achievable** for  $\kappa(\mathbf{u}) \geq v$  is based on a linear program  $\mathcal{L}_2(v)$ , which is defined over variables  $\mathcal{X}$  as in Eq. 9, and is obtained from  $\mathcal{L}_1(m)$  by replacing constraints (10c) with constraints (11c):

$\mathcal{L}_2(v) :$

max  $\xi$

subject to

$$\forall i \in [k] : \begin{cases} d(\alpha_i(s)) = 0, \\ d(s) \leq d(s') + \mathbf{c}[i](o), & \forall (s', a, s) \in G_i \\ \mathbf{b}[i] \leq d(s), & \forall s \in G_i \text{ s.t. } \mathbf{u}[i](s) = \sigma_i \end{cases}, \quad (11a)$$

$$\forall o \in O : \begin{cases} \mathbf{c}[i](o) \geq 0, & \forall i \in [k] \\ \sum_{i \in [k]} \mathbf{c}[i](o) \leq c(o) \end{cases}, \quad (11b)$$

$$\forall Z \subseteq [k] \text{ s.t. } \sum_{i \in Z} \sigma_i \geq v : \xi \leq \sum_{i \in Z} \mathbf{b}[i]. \quad (11c)$$

While the semantics of all variables but  $\xi$  remains as in  $\mathcal{L}_1(m)$ ,  $\xi$  now captures the minimal total cost of reaching some states  $\{s_i\}_{i \in [k]}$  in the abstract models  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}$  such that the total value  $\sum_{i \in [k]} \mathbf{u}[i](s_i) \geq v$ . The new constraint (11c) enforces this semantics of  $\xi$ .

**Lemma 8** *For any  $\epsilon > 0$ , the algorithm in Figure 9 terminates in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ ,  $\log \frac{1}{\epsilon}$ , and a unary representation of the budget  $b$  of  $\Pi$ .*

*Proof:* The number of iterations of the while-loop is approximately  $\log_2 \frac{\sum_{i \in [k]} \sigma_i}{\epsilon}$ , and the run-time of each of its iterations boils down to the complexity of solving  $\mathcal{L}_2(v)$ . Similarly to what we had in Lemma 5 with linear programs  $\mathcal{L}_1(m)$ , while the number of variables in  $\mathcal{L}_2(v)$ , as well as the number of constraints in (11a) and (11b), are polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ , the number of constraints in (11c) is  $\Theta(2^k)$ . Therefore,  $\text{solve}(\mathcal{L}_2(v))$  also employs the ellipsoid method with a sub-routine for the associated separation problem. We now show how that separation problem for  $\mathcal{L}_2(v)$  can be solved in pseudo-polynomial time using a standard pseudo-polynomial procedure for the *strict* Knapsack problem.

Given an assignment  $\mathbf{x} \in \text{dom}(\mathcal{X})$ , its feasibility with respect to (11a) and (11b) can be tested directly by substitution. For constraints (11c), let  $Z \subseteq [k]$  be an optimal solution to the strict Knapsack problem  $\langle \{\mathbf{x}[\mathbf{b}[i]], \sigma_i\}_{i \in [k]}, \mathbf{x}[\xi] \rangle$ , with a weight allowance  $\mathbf{x}[\xi]$  and  $k$  objects, with each object  $i \in [k]$  being associated with weight  $\mathbf{x}[\mathbf{b}[i]]$  and value  $\sigma_i$ .

- If the value  $\sum_{i \in Z} \sigma_i$  of  $Z$  is *smaller* than  $v$ , then  $\mathbf{x}$  satisfies *all* the constraints in (11c). Assume to the contrary that  $\mathbf{x}$  violates some constraint in (11c), corresponding to a set  $Z' \subseteq [k]$ . By definition of (11c),  $\sum_{i \in Z'} \sigma_i \geq v$ , and by our assumption,  $\mathbf{x}[\xi] > \sum_{i \in Z'} \mathbf{x}[\mathbf{b}[i]]$ . That, however, implies that  $Z'$  is a feasible solution for our strict Knapsack, and of value higher than that of presumably optimal  $Z$ .

- Otherwise, if  $\sum_{i \in Z} \sigma_i \geq v$ , then  $Z$  itself provides us with a constraint in (11c) that is violated by  $\mathbf{x}$ . This is because  $\mathbf{x}[\xi] > \sum_{i \in Z} \mathbf{x}[\mathbb{b}[i]]$  holds by the virtue of  $Z$  being a solution to the *strict* Knapsack problem  $\langle \{\mathbf{x}[\mathbb{b}[i]], \sigma_i\}_{i \in [k]}, \mathbf{x}[\xi] \rangle$ .

□

**Lemma 9** *For any  $0 < \epsilon < \min_{i \in [k]} \sigma_i$ , the algorithm in Figure 9a computes  $\kappa_\epsilon$  such that  $\kappa_\epsilon - \kappa(\mathbf{u}) \leq \epsilon$ .*

The proof of Lemma 9 appears in Appendix A, p. 47. Putting Lemmas 8 and 9 together, Theorem 10 summarizes our result for optimized abstraction discovery in  $\mathbf{A}_p(-, \mathbf{u}, -)$  for general 0-binary value partitions  $\mathbf{u}$ . Importantly, note that the algorithm in Figure 9 depends on the unary representation of only the budget, and not of the possible state values. In particular, it means that dependence of the complexity on the number of alternative sub-goals in the OSP task of interest is only polynomial. Finally, the statement of Theorem 10 involves the precision of the estimate only because the  $\sigma_i$  values of the abstract value functions  $\mathbf{u}[i]$  can be arbitrary real numbers. In the case of integer-valued sets of functions  $\mathbf{u}$ , as well as in various special cases of real-valued functions,  $\kappa(\mathbf{u})$  can be determined precisely using a simplification of the algorithm in Figure 9. For instance, if all  $\sigma_1, \dots, \sigma_k$  are integers, then setting  $\epsilon$  to any value in  $(0, 1)$  results in the while-loop terminating with  $\alpha = \kappa(\mathbf{u})$ . These details, however, are more of a theoretical interest; for reasonably small values of  $\epsilon$ , in practice there will be no difference between estimates  $h(s, b)$  and  $h(s, b) + \epsilon$ .

**Theorem 10 ( $\mathbf{A}_p(-, \mathbf{u}, -)(s)$  & 0-binary  $\mathbf{u}$ )**

*Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ , a 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$ , and  $\epsilon > 0$ , approximating  $\kappa_s(\mathbf{u})$  within an additive factor of  $\epsilon$  can be done in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ ,  $\log \frac{1}{\epsilon}$ , and a unary representation of the budget  $b$  of  $\Pi$ .*

## 4.2 General Value Partitions

While 0-binary value partitions can be rather useful by themselves, turns out that the pseudo-polynomial algorithms for abstraction discovery with explicit homomorphic abstraction skeletons and 0-binary value partitions can be extended rather easily to *arbitrary value partitions*. All it takes is to notice that,

- (1) For any OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , any homomorphic abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ , and any value partition  $\mathbf{u}$  over  $\mathcal{AS}$ , then number of distinct values taken by  $\mathbf{u}[i]$  is trivially upper-bounded by the number of states in  $G_i$ ; and
- (2) The pseudo-polynomial solvability of the Knapsack problem extends to its more general variant known as Multiple-Choice Knapsack (Dudzinski & Walukiewicz, 1987; Kellerer et al., 2004).

The **Multiple-Choice (MC) Knapsack** problem  $\langle N_1, \dots, N_m; W \rangle$  is given by a weight allowance  $W$  and  $m$  classes of objects  $N_1, \dots, N_m$ , with each object  $j \in N_i$  being annotated

with a weight  $w_{ij}$  and a value  $\sigma_{ij}$ . The objective is to find a set  $Z$  that contains at most one object from each class and maximizes  $\sum_{(i,j) \in Z} \sigma_{ij}$  over all such sets while satisfying  $\sum_{(i,j) \in Z} w_{ij} \leq W$ . By **strict MC-Knapsack** we refer to a variant of MC-Knapsack in which that inequality constraint is strict. MC-Knapsack generalizes regular Knapsack and thus it is NP-hard. However, similarly to the regular Knapsack, MC-Knapsack also admits a pseudo-polynomial, dynamic programming algorithm that runs in time polynomial in the description of the problem and in the unary representation of  $W$  (Dudzinski & Walukiewicz, 1987; Kellerer et al., 2004).

**Theorem 11 ( $\mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$ )**

Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  be an explicit homomorphic abstraction skeleton of  $M_\Pi$ , and  $\mathbf{u} \in \mathbf{U}_p$  be an arbitrary value partition over  $\mathcal{AS}$ . Given a cost partition  $\mathbf{c} \in \mathbf{C}_p$ , finding an abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p(\mathbf{c}, \mathbf{u}, -)$  and computing the corresponding heuristic estimate  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0, b)$  can be done in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ , and the unary representation of the budget  $b$ .

*Proof:* The proof is very similar to the proof of Theorem 3, but with the compilation being to the MC-Knapsack problem.

For  $i \in [k]$ , let  $\mathbf{u}[i]$  be to  $\{\sigma_{i1}, \dots, \sigma_{in_i}\} \subset \mathbb{R}^+$ , and, for  $j \in [n_i]$ ,  $w_{ij}$  be the cost of the cheapest path in  $G_i$  from  $\alpha_i(s_0)$  to (one of the) states  $s \in S_i$  with  $\mathbf{u}[i](s) = \sigma_{ij}$ . Since  $\mathcal{AS}$  is an *explicit* abstraction skeleton, for  $i \in [k]$ ,  $n_i \leq |S_i|$ , and the set  $\{w_{ij}\}_{i \in [k], j \in [n_i]}$  can be computed in time polynomial in  $\|\mathcal{AS}\|$  using one of the standard algorithms for the single-source shortest paths problem.

Consider now an MC-Knapsack problem with a weight allowance  $b$  and  $k$  classes of objects  $N_1, \dots, N_k$ , with  $|N_i| = n_i$  and each object  $j \in N_i$  being annotated with a weight  $w_{ij}$  and a value  $\sigma_{ij}$ . Let  $Z \subseteq \bigcup_{i=1}^k N_i$  be a solution to that (optimization) MC-Knapsack problem; recall that it is computable in pseudo-polynomial time. Given that, we define budget profile  $\mathbf{b}^* \in \mathbf{B}$  as follows:

$$\text{for } i \in [k], \mathbf{b}^*[i] = \begin{cases} w_{ij}, & (i, j) \in Z \\ 0, & \text{otherwise.} \end{cases}$$

Showing that  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  actually induces an additive abstraction for  $M_\Pi$  is completely identical to the proof of the corresponding argument in Theorem 3, and thus omitted.  $\square$

**Theorem 12 ( $\mathbf{A}_p(-, \mathbf{u}, -)$ )**

Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ , an arbitrary value partition  $\mathbf{u} \in \mathbf{U}_p$  over  $\mathcal{AS}$ , and  $\epsilon > 0$ , approximating  $\kappa_s(\mathbf{u})$  within an additive factor of  $\epsilon$  can be done in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ ,  $\log \frac{1}{\epsilon}$ , and a unary representation of the budget  $b$  of  $\Pi$ .

An algorithm for abstraction discovery as in Theorem 12 is depicted in Figure 10. Its high-level flow differs from the flow of the algorithm from Figure 9 for general 0-binary value partitions only in the initialization of parameters  $\epsilon$  and  $\beta$ . The major difference between the algorithms is that here the tests of candidate values  $v$  are delegated to a different



input:  $\Pi = \langle V, s_0, u; O, c, b \rangle$ ,  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ ,  
 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$   
 output:  $\kappa_s(\mathbf{u})$

```

for  $i = 1$  to  $k$  do
  reduce  $G_i$  to only nodes reachable from  $\alpha_i(s_0)$ 
let  $0 < \epsilon < \min_{i \in [k]} \min_{j \in [n_i]} \sigma_{ij}$ 
 $\alpha \leftarrow 0$ 
 $\beta \leftarrow \sum_{i \in [k]} \max_{j \in [n_i]} \sigma_{ij}$ 
while  $\beta - \alpha > \epsilon$  do
   $v \leftarrow \alpha + (\beta - \alpha)/2$ 
  if  $\text{always-achievable}(v)$  then  $\alpha \leftarrow v$ 
  else  $\beta \leftarrow v$ 
if  $\alpha = 0$  then return 0
else return  $\beta$ 

```

```

always-achievable( $v$ ):
  solve( $\mathcal{L}_2(v)$ )  $\mapsto$  solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$ 
  if  $\mathbf{x}[\xi] \leq b$  then return true
  else return false

```

(a)

```

solve( $\mathcal{L}_3(v)$ ):
  set (13c') to an arbitrary subset of constraints (13c)
  loop
    set  $\mathcal{L}'_3(v)$  to  $\mathcal{L}_3(v)$ , with constraints (13c') instead of (13c)
    ellipsoid-method( $\mathcal{L}'_3(v)$ )  $\mapsto$  solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$ 
    strict-MC-Knapsack( $\langle \{\mathbf{x}[\mathbb{b}[1, j]], \sigma_{1j}\}_{j \in [n_1]}, \dots, \{\mathbf{x}[\mathbb{b}[k, j]], \sigma_{kj}\}_{j \in [n_k]}; \mathbf{x}[\xi] \rangle$ )
       $\mapsto$  solution  $Z \in [n_1] \times \dots \times [n_k]$ 
    if  $\sum_{i \in [k]} \sigma_{iZ(i)} < v$  then return  $\mathbf{x}$ 
  extend (13c') with constraint  $\xi \leq \sum_{i \in [k]} \mathbb{b}[i, Z(i)]$ 

```

(b)

Figure 10: (a) A modification of the algorithm from Figure 9 to arbitrary value partitions  $\mathbf{u} \in \mathbf{U}_p$  (Theorem 12), and (b) the respective solve sub-routine that is based on linear programs  $\mathcal{L}_3(v)$  in Eq. 13

solve sub-routine (Figure 10b) that is based on linear programs  $\mathcal{L}_3(v)$ , which are defined as follows.

For  $i \in [k]$ , let  $\mathbf{u}[i]$  be to  $\{\sigma_{i1}, \dots, \sigma_{in_i}\} \subset \mathbb{R}^+$ . For  $v \in \mathbb{R}^+$ , the linear program  $\mathcal{L}_3(v)$  is defined in Eq. 13 over variables

$$\mathcal{X} = \{\xi\} \cup \bigcup_{i \in [k]} \left[ \{d(s)\}_{s \in G_i} \cup \bigcup_{j \in [n_i]} \{\mathbb{b}[i, j]\} \cup \bigcup_{o \in O} \{\mathbb{c}[i](o)\} \right]. \quad (12)$$

These variables differ from the variable set of  $\mathcal{L}_2(v)$  (see Eq. 9) by a larger set of  $\mathbb{b}$ -variables: Variable  $\mathbb{b}[i, j]$  here captures the minimal budget needed for reaching in  $G_i$  a state with value  $\sigma_{i,j}$  from state  $\alpha_i(s_0)$ , given that the edges of  $G_i$  are weighted consistently with the variable vector  $\mathbb{c}[i]$ .

$$\begin{aligned} &\mathcal{L}_3(v) : \\ &\quad \max \xi \\ &\quad \text{subject to} \\ &\quad \forall i \in [k] : \begin{cases} d(\alpha_i(s)) = 0, \\ d(s) \leq d(s') + \mathbb{c}[i](o), & \forall (s', a, s) \in G_i \\ \mathbb{b}[i, j] \leq d(s), & \forall j \in [n_i] \forall s \in G_i \text{ s.t. } \mathbf{u}[i](s) = \sigma_{ij} \end{cases}, \end{aligned} \quad (13a)$$

$$\forall o \in O : \begin{cases} \mathbb{c}[i](o) \geq 0, & \forall i \in [k] \\ \sum_{i \in [k]} \mathbb{c}[i](o) \leq c(o) \end{cases}, \quad (13b)$$

$$\begin{aligned} &\forall Z \in [n_1] \times \dots \times [n_k] \\ &\quad \text{s.t. } \sum_{i \in [k]} \sigma_{iZ(i)} \geq v : \xi \leq \sum_{i \in [k]} \mathbb{b}[i, Z(i)]. \end{aligned} \quad (13c)$$

Similarly to what we had in Lemma 8 with linear programs  $\mathcal{L}_2(v)$ , while the number of variables in  $\mathcal{L}_3(v)$ , as well as the number of constraints in (13a) and (13b), are polynomial in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ , the number of constraints in (13c) is  $\Theta(d^k)$  where  $d = \max_{i \in [k]} n_i$ . Therefore,  $\text{solve}(\mathcal{L}_3(v))$  also employs the ellipsoid method with a pseudo-polynomial separation problem, but here it is based on strict MC-Knapsack. Otherwise, solving  $\mathcal{L}_2(v)$  and solving  $\mathcal{L}_3(v)$  are similar.

**Lemma 13** *For any  $\epsilon > 0$ , the algorithm in Figure 10 terminates in time polynomial in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ ,  $\log \frac{1}{\epsilon}$ , and a unary representation of the budget  $b$  of  $\Pi$ .*

**Lemma 14** *Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ , an arbitrary value partition  $\mathbf{u} \in \mathbf{U}_p$  over  $\mathcal{AS}$ , and  $\epsilon > 0$ , the algorithm in Figure 10 computes  $\kappa_\epsilon$  such that  $\kappa_\epsilon - \kappa(\mathbf{u}) \leq \epsilon$ .*

The proof of Lemma 13 is similar to the proof of Lemma 8, with strict Knapsack separation problems being replaced with strict MC-Knapsack separation problems. The proof of Lemma 14 is also similar to the proof of Lemma 9, *mutatis mutandis*. Together, Lemmas 14 and 13 establish Theorem 12.

## 5. Landmarks in OSP

In addition to state-space abstractions, a family of approximation techniques that have been found extremely effective in the context of optimal classical planning is based on the notion of logical *landmarks for goal reachability* (Karpas & Domshlak, 2009; Helmert & Domshlak, 2009; Domshlak et al., 2012; Bonet & Helmert, 2010b; Pommerening & Helmert, 2013). In this section we proceed with examining the prospects of such reachability landmarks in heuristic-search OSP planning.

### 5.1 Landmarks in Classical Planning

For a state  $s$  in a classical planning task  $\Pi$ , a landmark is a property of operator sequences that is satisfied by all  $s$ -plans (Hoffmann, Porteous, & Sebastia, 2004). For instance, a “fact landmark” for a state  $s$  is an assignment to a single variable that is true at some point in *every*  $s$ -plan. Most state-of-the-art admissible heuristics for classical planning use what is called **disjunctive action landmarks**, each corresponding to set of operators such that every  $s$ -plan contains at least one operator from that set (Karpas & Domshlak, 2009; Helmert & Domshlak, 2009; Bonet & Helmert, 2010a; Pommerening & Helmert, 2013). In what follows we consider this popular notion of landmarks, and simply refer to disjunctive action landmarks for a state  $s$  as *s-landmarks*. For ease of presentation, most of our discussion will take place in the context of landmarks for the initial state of the task, and these will simply be referred to as *landmarks (for  $\Pi$ )*.

Deciding whether an operator set  $L \subset O$  is a landmark for classical planning task  $\Pi$  is PSPACE-hard (Porteous, Sebastia, & Hoffmann, 2001). Therefore, all landmark heuristics employ methods for landmark discovery that are polynomial-time, sound, but incomplete. In what follows we assume access to such a procedure; the actual way the landmarks are discovered is tangential to our contribution. For a set  $\mathcal{L}$  of  $s$ -landmarks, a **landmark cost function**  $lcost : \mathcal{L} \rightarrow \mathbb{R}^{0+}$  is *admissible* if  $\sum_{L \in \mathcal{L}} lcost(L) \leq h^*(s)$ . For a singleton set  $\mathcal{L} = \{L\}$ ,  $lcost(L) := \min_{o \in L} c(o)$  is a natural admissible landmark cost function, and it extends directly to non-singleton sets of pairwise disjoint landmarks. For more general sets of landmarks,  $lcost$  can be devised in polynomial time via operator cost partitioning (Katz & Domshlak, 2010b), either given  $\mathcal{L}$  (Karpas & Domshlak, 2009), or within the actual process of generating  $\mathcal{L}$  (Helmert & Domshlak, 2009).

### 5.2 $\epsilon$ -Landmarks and Budget Reduction

While landmarks play an important role in (both satisficing and optimal) classical planning, so far they have not been exploited in OSP. At first glance, this is probably no surprise, and not only because OSP have been investigated much less than classical planning: Since landmarks must be satisfied by all plans, and empty operator sequence is always a plan for any OSP task, the notion of landmark does not seem useful here. Having said that, consider the anytime “output improvement” property of the *BFBB* forward search. The empty plan is not interesting there not only because it is useless, but also because it is “found” by the search algorithm right at the get-go. In general, at all stages of the search, anytime algorithms like *BFBB* maintain the best-so-far solution  $\pi$ , and prune all branches that promise value lower or equal to  $Q^b(\pi)$ . Hence, in principle, such algorithms may benefit

from information about properties that are “satisfied by all plans with value larger than  $Q^b(\pi)$ .” Polynomial-time discovery of such “value landmarks” for arbitrary OSP tasks is still an open problem. However, looking at what is needed and what is available, here we show that classical planning machinery of reachability landmarks actually can be effectively exploited in OSP.

In what follows, we assume that the value function of  $\Pi$  is *additive*, with  $u(s) = \sum_{\langle v/d \rangle \in s} u_v(d)$ , with  $u_v(d) \geq 0$  for all variable-value pairs  $\langle v/d \rangle$ . That is, the value of state  $s$  is the sum of the (mutually independent) non-negative marginal values of the propositions comprising  $s$ . With the value of different  $s$ -plans in an OSP task  $\Pi$  varying between zero and the value of the optimal  $s$ -plan (which may also be zero), let  $\varepsilon$ -**landmark** for state  $s$  be any property that is satisfied by any  $s$ -plan  $\pi$  that achieves *something valuable*. For instance, with the disjunctive action landmarks we use here, if  $L \subseteq O$  is an  $\varepsilon$ -landmark for  $s$ , then every  $s$ -plan  $\pi$  with  $Q^b(\pi) > 0$  contains an operator from  $L$ . In what follows, unless stated otherwise, we focus on  $\varepsilon$ -landmarks for (the initial state of)  $\Pi$ .

**Definition 5** *Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$ , the  $\varepsilon$ -**compilation** of  $\Pi$  is a classical planning task  $\Pi_\varepsilon = \langle V_\varepsilon, s_{0_\varepsilon}, G_\varepsilon; O_\varepsilon, c_\varepsilon \rangle$  where*

$$\begin{aligned} V_\varepsilon &= V \cup \{g\}, \\ &\text{with } \text{dom}(g) = \{0, 1\}, \\ s_{0_\varepsilon} &= s_0 \cup \{\langle g/0 \rangle\}, \\ G_\varepsilon &= \{\langle g/1 \rangle\}, \\ O_\varepsilon &= O \cup O_g = O \cup \{o_{\langle v/d \rangle} \mid \langle v/d \rangle \in \mathcal{D}, u_v(d) > 0\}, \\ &\text{with } \text{pre}(o_{\langle v/d \rangle}) = \{\langle v/d \rangle\} \text{ and } \text{eff}(o_{\langle v/d \rangle}) = \{\langle g/1 \rangle\}, \\ c_\varepsilon(o) &= \begin{cases} c(o), & \omega = o \in O \\ 0, & \omega = o_{\langle v/d \rangle} \in O_g \end{cases}. \end{aligned}$$

In plain words,  $\Pi_\varepsilon$  extends the structure of  $\Pi$  with a set of zero-cost actions such that applying any of them indicates achieving a positive value in  $\Pi$ . Constructing  $\Pi_\varepsilon$  from  $\Pi$  is trivially polynomial time, and it allows us to discover  $\varepsilon$ -landmarks for  $\Pi$  using the standard machinery for classical planning landmark discovery.

**Theorem 15** *For any OSP task  $\Pi$ , any landmark  $L$  for  $\Pi_\varepsilon$  such that  $L \subseteq O$  is an  $\varepsilon$ -landmark for  $\Pi$ .*

*Proof:* The proof is rather straightforward. Let  $\mathcal{P}$  be the set of all plans  $\pi$  for  $\Pi$  with  $Q^b(\pi) > 0$  and  $\mathcal{P}_\varepsilon$  the set of all plans for  $\Pi_\varepsilon$ . By the definition of  $\mathcal{P}$ , for any plan  $\pi \in \mathcal{P}$ , there exists a proposition  $\langle v/d \rangle \in \mathcal{D}$  such that  $u_v(d) > 0$  and  $\langle v/d \rangle \in s_0[\pi]$ . Likewise, since  $s_{0_\varepsilon} := s_0 \cup \{\langle g/0 \rangle\}$  and  $O_\varepsilon \supseteq O$ ,  $\pi$  is applicable in  $s_{0_\varepsilon}$ . Hence, by definition of  $o_{\langle v/d \rangle} \in O_\varepsilon$ ,  $\pi \cdot \langle o_{\langle v/d \rangle} \rangle$  is applicable in  $s_{0_\varepsilon}$  and  $\langle g/1 \rangle \in s_{0_\varepsilon}[\pi \cdot \langle o_{\langle v/d \rangle} \rangle]$ , that is,  $\pi \cdot \langle o_{\langle v/d \rangle} \rangle \in \mathcal{P}_\varepsilon$ . In turn, if  $L$  is a landmark for  $\Pi_\varepsilon$ , then  $\pi \cdot \langle o_{\langle v/d \rangle} \rangle$  contains an operator from  $L$ , and if  $L \subseteq O$ , then  $\pi$  contains an operator from  $L$  as well. That proves that all landmarks  $L$  for  $\Pi_\varepsilon$  over the operators  $O$  of  $\Pi$  are  $\varepsilon$ -landmarks for  $\Pi$ .  $\square$

With Theorem 15 in hand, we can now derive  $\varepsilon$ -landmarks for  $\Pi$  using any method for classical planning landmark extraction, such as that employed by the LAMA planner (Richter et al., 2008) or the LM-Cut family of techniques (Helmert & Domshlak, 2009; Bonet & Helmert, 2010a). However, at first glance, the discriminative power of knowing “what is needed to achieve *something* valuable” seems to be negligible when it comes to deriving effective heuristic estimates for OSP. The good news is that, in OSP, such information can be effectively exploited in a slightly different way.

Consider a schematic example of searching for an optimal plan for an OPS task  $\Pi$  with budget  $b$ , using *BFBB* with an admissible heuristic  $h$ . Suppose that there is only one sequence of (all unit-cost) operators,  $\pi = \langle o_1, o_2, \dots, o_{b+1} \rangle$ , applicable in the initial state of  $\Pi$ , and that the only positive value state along  $\pi$  is its end-state. While clearly no value higher than zero can be achieved in  $\Pi$  under the given budget of  $b$ , the search will continue beyond the initial state, unless  $h(s_0, \cdot)$  counts the cost of all the  $b + 1$  operators of  $\pi$ . Now, suppose that  $h(s_0, \cdot)$  counts only the cost of  $\{o_i, \dots, o_{b+1}\}$  for some  $i > 0$ , but  $\{o_1\}, \{o_2\}, \dots, \{o_{i-1}\}$  are all discovered to be  $\varepsilon$ -landmarks for  $\Pi$ . Given that, suppose that we modify  $\Pi$  by (a) setting the cost of operators  $o_1, o_2, \dots, o_{i-1}$  to zero, and (b) *reducing the budget* to  $b - i + 1$ . Since all the operators  $o_1, o_2, \dots, o_{i-1}$  anyway have to be applied along any value collecting plan for  $\Pi$ , this modification seems to preserve the semantics of  $\Pi$ . At the same time, on the modified task, *BFBB* with the same heuristic  $h$  will prune the initial state and thus establish without any search that the empty plan is an optimal plan for  $\Pi$ . Of course, the way  $\Pi$  is modified in this example is as simplistic as the example itself. Yet, this example does motivate the idea of *landmark-based budget reduction* for OSP, as well as illustrates the basic idea behind the *generically sound* task modifications that we discuss next.

**Definition 6** Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ , and  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ . The **budget reducing compilation** of  $\Pi$  is an OSP task  $\Pi_{\mathcal{L}} = \langle V_{\mathcal{L}}, s_{0\mathcal{L}}, u_{\mathcal{L}}; O_{\mathcal{L}}, c_{\mathcal{L}}, b_{\mathcal{L}} \rangle$  where

$$b_{\mathcal{L}} = b - \sum_{i=1}^n lcost(L_i) \quad (14)$$

and

$$\begin{aligned} V_{\mathcal{L}} &= V \cup \{v_{L_1}, \dots, v_{L_n}\} \\ &\quad \text{with } dom(v_{L_i}) = \{0, 1\}, \\ s_{0\mathcal{L}} &= s_0 \cup \{\langle v_{L_1}/1 \rangle, \dots, \langle v_{L_n}/1 \rangle\}, \\ u_{\mathcal{L}} &= u, \\ O_{\mathcal{L}} &= O \cup \bigcup_{i=1}^n O_{L_i} = O \cup \bigcup_{i=1}^n \{\bar{o} \mid o \in L_i\}, \\ &\quad \text{with } pre(\bar{o}) = pre(o) \cup \{\langle v_{L_i}/1 \rangle\} \text{ and } eff(\bar{o}) = eff(o) \cup \{\langle v_{L_i}/0 \rangle\}, \\ c_{\mathcal{L}}(\omega) &= \begin{cases} c(o), & \omega = o \in O \\ c(o) - lcost(L_i), & \omega = \bar{o} \in O_{L_i} \end{cases}. \end{aligned}$$

**compile-and-BFBB** ( $\Pi = \langle V, s_0, u; O, c, b \rangle$ )  
 $\Pi_\varepsilon := \varepsilon$ -compilation of  $\Pi$   
 $\mathcal{L} :=$  a set of landmarks for  $\Pi_\varepsilon$   
 $lcost :=$  admissible landmark cost function from  $\mathcal{L}$   
 $\Pi_{\mathcal{L}}^* :=$  budget reducing compilation of  $(\mathcal{L}, lcost)$  into  $\Pi$   
 $n^* := \text{BFBB}(\Pi_{\mathcal{L}}^*)$   
**return** plan for  $\Pi$  associated with  $n^*$

Figure 11: *BFBB* search with landmark-based budget reduction

In other words,  $\Pi_{\mathcal{L}}$  extends the structure of  $\Pi$  by

- mirroring the operators of each  $\varepsilon$ -landmark  $L_i$  with their “cheaper by  $lcost(L_i)$ ” versions,
- using the “disposable” propositions  $\langle v_{L_1}/1 \rangle, \dots, \langle v_{L_n}/1 \rangle$  to ensure that at most one instance of these discounted operators for each  $L_i$  can be applied along an operator sequence from the initial state, and
- compensating for the discounted operators for  $L_i$  by reducing the budget by precisely  $lcost(L_i)$ .

This way, the transformation leads to effective equivalence between  $\Pi$  and  $\Pi_{\mathcal{L}}$ .

**Theorem 16** *Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ ,  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}}$  be the respective budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $Q^b(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}}$  for  $\Pi_{\mathcal{L}}$  with  $Q^{b_{\mathcal{L}}}(\pi_{\mathcal{L}}) = Q^b(\pi)$ , and vice versa.*

The proof of Theorem 16 appears in Appendix A, p. 48. The budget reducing OSP-to-OSP compilation in Definition 6 is clearly polynomial time. Putting things together, the **compile-and-BFBB** procedure depicted in Figure 11:

- (1) generates an  $\varepsilon$ -compilation  $\Pi_\varepsilon$  of  $\Pi$ ,
- (2) uses off-the-shelf tools for classical planning to generate a set of landmarks  $\mathcal{L}$  for  $\Pi_\varepsilon$  and an admissible landmark cost function  $lcost$ , and
- (3) compiles  $(\mathcal{L}, lcost)$  into  $\Pi$ , obtaining an OSP task  $\Pi_{\mathcal{L}}$ .

The optimal solution for  $\Pi_{\mathcal{L}}$  (and thus for  $\Pi$ ) is then searched for using a search algorithm for optimal OSP such as *BFBB*.

Before we proceed to consider more general sets of landmarks, a few comments concerning the setup of Theorem 16 are now probably in place. First, if the reduced budget  $b_{\mathcal{L}}$  turns out to be lower than the cost of the cheapest action applicable in the initial state, then obviously no search is needed, and the empty plan can be reported as optimal right away. Second, zero-cost landmarks are useless in our compilation as much as they are useless in deriving landmark heuristics for optimal planning. Hence,  $lcost$  in what follows is assumed

to be strictly positive. Third, having both  $o$  and  $\bar{o}$  applicable at a state of  $\Pi_\epsilon$  brings no benefits yet adds branching to the search. Hence, in our implementation, for each landmark  $L_i \in \mathcal{L}$  and each operator  $o \in L_i$ , the precondition of the regular operators  $o$  in  $O_{\mathcal{L}}$  is extended with  $\{\langle v_{L_i}/0 \rangle\}$ . It is not hard to verify that this extension preserves the correctness of  $\Pi_{\mathcal{L}}$  in terms of Theorem 16. Finally, if the value of the initial state is not zero, that is, the empty plan has some positive value, then  $\epsilon$ -compilation  $\Pi_\epsilon$  of  $\Pi$  will have no positive cost landmarks at all. However, this can easily be fixed by considering as “valuable” only propositions  $\langle v/d \rangle$  such that both  $u_v(d) > 0$  and  $\langle v/d \rangle \notin s_0$ . For now we put this issue aside and assume that  $Q^b(\epsilon) = 0$ . Later, however, we come back to consider this issue more systematically.

### 5.3 Non-Disjoint $\epsilon$ -Landmarks

While the budget reducing compilation  $\Pi_{\mathcal{L}}$  above is sound for pairwise disjoint landmarks, this is not so for more general sets of  $\epsilon$ -landmarks. For example, consider a planning task  $\Pi$  in which, for some operator  $o$ , we have  $c(o) = b$ ,  $Q^b(\langle o \rangle) > 0$ , and  $Q^b(\pi) = 0$  for all other operator sequences  $\pi \neq \langle o \rangle$ . That is, a value greater than zero is achievable in  $\Pi$ , but only via the operator  $o$ . Suppose now that our set of  $\epsilon$ -landmarks for  $\Pi$  is  $\mathcal{L} = \{L_1, \dots, L_n\}$ ,  $n > 1$ , and that all of these  $\epsilon$ -landmarks contain  $o$ . In this case, while the budget in  $\Pi_{\mathcal{L}}$  is  $b_{\mathcal{L}} = b - \sum_{i=1}^n lcost(L_i)$ , the cost of the cheapest replica  $\bar{o}$  of  $o$ , that is, the cost of the cheapest operator sequence achieving a non-zero value in  $\Pi$ , is

$$c(o) - \min_{i=1}^n lcost(L_i) = b - \min_{i=1}^n lcost(L_i) > b - \sum_{i=1}^n lcost(L_i) = b_{\mathcal{L}}.$$

Hence, no state with positive value will be reachable from  $s_{0\mathcal{L}}$  in  $\Pi_{\mathcal{L}}$ , and thus  $\Pi$  and  $\Pi_{\mathcal{L}}$  are not “value equivalent” in the sense of Theorem 16.

This example shows why compiling non-disjoint  $\epsilon$ -landmarks into  $\Pi$  independently is not sound. In principle, this can be repaired as follows. Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of  $\epsilon$ -landmarks for  $\Pi$ , and  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ . All the components in  $\Pi_{\mathcal{L}} = \langle V_{\mathcal{L}}, s_{0\mathcal{L}}, u_{\mathcal{L}}; O_{\mathcal{L}}, c_{\mathcal{L}}, b_{\mathcal{L}} \rangle$  are still defined as in Definition 6, except for the operator sets  $O_{L_1}, \dots, O_{L_n}$ . The latter are now constructed not independently of each other, but sequentially, with the content of  $O_{L_i}$  depending on the content of all  $O_{L_j}$ ,  $j < i$ . The ordering in which the sets  $O_{L_i}$  are constructed can be arbitrary.

For each operator  $o \in O$  and each  $1 \leq i \leq n$ , let  $\bar{O}_{o';i}$  denote the set of all “cost discounted” representatives of  $o$  introduced during the construction of  $O_{L_1}, \dots, O_{L_i}$ . Given that, for  $1 \leq i \leq n$ , if for some operator  $o \in \bigcup_{o' \in L_i} \bar{O}_{o';i-1}$ , we have  $c_{\mathcal{L}}(o) = 0$ , then  $O_{L_i} := \emptyset$ . Otherwise,  $O_{L_i}$  contains an operator  $\bar{o}$  for each operator

$$o \in L_i \cup \bigcup_{o' \in L_i} \bar{O}_{o';i-1}, \quad (15)$$

with  $\bar{o}$  being defined very similarly to Definition 6 as:

$$\begin{aligned} \text{pre}(\bar{o}) &= \text{pre}(o) \cup \{\langle v_{L_i}/1 \rangle\}, \\ \text{eff}(\bar{o}) &= \text{eff}(o) \cup \{\langle v_{L_i}/0 \rangle\}, \\ c_{\mathcal{L}}(\bar{o}) &= \begin{cases} c(o) - \text{lcost}(L_i), & o \in L_i, \\ c_{\mathcal{L}}(o) - \text{lcost}(L_i), & o \in \bigcup_{o' \in L_i} \bar{O}_{o'; i-1} \end{cases}. \end{aligned} \quad (16)$$

The compilation extended this way is sound for arbitrary sets of  $\varepsilon$ -landmarks, and on pairwise disjoint landmarks it reduces to the basic compilation used in Theorem 16. In general, however, this extended compilation is no longer polynomial in the size of the explicit representation of  $\Pi$  because

$$|\bar{O}_{o;i}| = 2^{|\{L_j | j \leq i, o \in L_j\}|}.$$

For example, let  $\mathcal{L} = \{L_1, L_2, L_3\}$ ,  $L_1 = \{a, b\}$ ,  $L_2 = \{a, c\}$ ,  $L_3 = \{a, d\}$ . Generation of  $O_{L_1} := \{\bar{a}_1, \bar{b}_1\}$  effectively follows Definition 6, but for  $O_{L_2}$ , the base set of operators as in Eq. 15 is already  $\{a, c, \bar{a}_1\}$ . Thus,  $O_{L_2} := \{\bar{a}_2, \bar{c}_1, \bar{a}_3\}$ , where, for  $i \in \{2, 3\}$  and denoting  $a$  by  $\bar{a}_0$ ,  $\bar{a}_i$  is derived according to Eq. 16 from  $\bar{a}_{i-2}$ . Consequently, the base set of operators for  $O_{L_3}$  is  $\{a, d, \bar{a}_1, \bar{a}_2, \bar{a}_3\}$ , resulting in  $O_{L_3} = \{\bar{a}_4, \bar{d}_1, \bar{a}_5, \bar{a}_6, \bar{a}_7\}$ , where, for  $i \in \{4, 5, 6, 7\}$ ,  $\bar{a}_i$  is derived from  $\bar{a}_{i-4}$ . In sum,  $\Pi_{\mathcal{L}}$  ends up with  $8 = 2^{|\mathcal{L}|}$  representatives of the operator  $a$ .

Since non-disjoint landmarks can bring more information, and they are typical to outputs of standard techniques for landmark extraction in classical planning, we now present a different, slightly more involved, compilation that is both polynomial and sound for arbitrary sets of  $\varepsilon$ -landmarks.

**Definition 7** Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ , and  $\text{lcost}$  be an admissible landmark cost function from  $\mathcal{L}$ . For each operator  $o$ , let  $\mathcal{L}(o)$  denote the set of all landmarks in  $\mathcal{L}$  that contain  $o$ . Given that, the **generalized budget reducing compilation** of  $\Pi$  is an OSP task  $\Pi_{\mathcal{L}^*} = \langle V_{\mathcal{L}^*}, s_{0\mathcal{L}^*}, u_{\mathcal{L}^*}; O_{\mathcal{L}^*}, c_{\mathcal{L}^*}, b_{\mathcal{L}^*} \rangle$  where and

$$\begin{aligned} b_{\mathcal{L}^*} &= b - \sum_{i=1}^n \text{lcost}(L_i) \\ V_{\mathcal{L}^*} &= V \cup \{v_{L_1}, \dots, v_{L_n}\} \\ &\quad \text{with } \text{dom}(v_{L_i}) = \{0, 1\}, \\ s_{0\mathcal{L}^*} &= s_0 \cup \{\langle v_{L_1}/1 \rangle, \dots, \langle v_{L_n}/1 \rangle\}, \\ u_{\mathcal{L}^*} &= u, \\ O_{\mathcal{L}^*} &= O \cup \{\bar{o} \mid o \in \bigcup_{L \in \mathcal{L}} L\} \cup \{\text{get}(L) \mid L \in \mathcal{L}\} \end{aligned}$$

with

$$\begin{aligned} \text{pre}(\bar{o}) &= \text{pre}(o) \cup \{\langle v_L/1 \rangle \mid L \in \mathcal{L}(o)\}, \\ \text{eff}(\bar{o}) &= \text{eff}(o) \cup \{\langle v_L/0 \rangle \mid L \in \mathcal{L}(o)\}, \end{aligned} \quad (17)$$

and

$$\begin{aligned} \text{pre}(\text{get}(L)) &= \{\langle v_L/0 \rangle\}, \\ \text{eff}(\text{get}(L)) &= \{\langle v_L/1 \rangle\}, \end{aligned} \quad (18)$$



and

$$c_{\mathcal{L}^*}(\omega) = \begin{cases} c(o), & \omega = o \in O \\ c(o) - \sum_{L \in \mathcal{L}(o)} lcost(L), & \omega = \bar{o} \\ lcost(L), & \omega = get(L) \end{cases}. \quad (19)$$

Illustrating this compilation, let  $\mathcal{L} = \{L_1, L_2, L_3\}$ ,

$$L_1 = \{a, b\},$$

$$L_2 = \{b, c\},$$

$$L_3 = \{a, c\},$$

with all operators having the cost of 2, and let

$$lcost(L_1) = lcost(L_2) = lcost(L_3) = 1.$$

In  $\Pi_{\mathcal{L}^*}$ , we have  $V_{\mathcal{L}^*} = V \cup \{v_{L_1}, v_{L_2}, v_{L_3}\}$  and

$$O_{\mathcal{L}^*} = O \cup \{\bar{a}, \bar{b}, \bar{c}, get(L_1), get(L_2), get(L_3)\},$$

with, e.g.,

$$\text{pre}(\bar{a}) = \text{pre}(a) \cup \{\langle v_{L_1}/1 \rangle, \langle v_{L_3}/1 \rangle\},$$

$$\text{eff}(\bar{a}) = \text{eff}(a) \cup \{\langle v_{L_1}/0 \rangle, \langle v_{L_3}/0 \rangle\},$$

$$c_{\mathcal{L}^*}(\bar{a}) = 0,$$

and, for  $get(L_1)$ ,

$$\text{pre}(get(L_1)) = \{\langle v_L/0 \rangle\},$$

$$\text{eff}(get(L_1)) = \{\langle v_L/1 \rangle\},$$

$$c_{\mathcal{L}^*}(get(L_1)) = 1.$$

The intuition behind the compilation in Definition 7 is as follows. By Eq. 19, applying a discounted operator  $\bar{o}$  saves the total cost of all landmarks containing  $o$ . Therefore,

- $\bar{o}$  is allowed to be executed only at states  $s$  in which all the corresponding control propositions  $\{\langle v_L/1 \rangle \mid L \in \mathcal{L}(o)\}$  hold, indicating that the cost of no landmark in  $\mathcal{L}(o)$  has already been saved before reaching  $s$ , and
- to avoid double savings around  $\mathcal{L}(o)$ , applying  $\bar{o}$  in  $s$  turns off all these control propositions in  $s[\![\bar{o}]\!]$ .

However, considering the example above, suppose that the optimal plan  $\pi$  for the original task contains an instance of operator  $a$ , followed by an instance of operator  $b$ , and no instance of operator  $c$ . Applying  $\bar{a}$  instead of  $a$  would block us from applying  $\bar{b}$  instead of  $b$ , and thus the value of the optimal plan in the compilation can be lower than  $Q^b(\pi)$ . The rescue here comes from the  $get(L)$  actions that allow for *selective* “wasting” of the *individual* landmark costs  $lcost(L)$ . In our example, while applying  $\bar{a}$  in  $s$  saves the cost of the landmarks  $L_1$  and  $L_3$ , applying then  $get(L_1)$  will waste  $lcost(L_1)$  and safely set the control proposition

$\langle v_{L_1}/1 \rangle$ . In turn, this will enable applying  $\bar{b}$  at the next steps, and applying  $\bar{b}$  will then save the cost of  $L_2$  and “re-save” the cost of  $L_1$ . This way, the compilation leads to effective equivalence between  $\Pi$  and  $\Pi_{\mathcal{L}^*}$ , which is formulated in Theorem 17 below, and proven in Appendix A, p. 49.

**Theorem 17** *Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of  $\varepsilon$ -landmarks for  $\Pi$ ,  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}^*}$  be the (generalized) budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $Q^b(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}^*}$  for  $\Pi_{\mathcal{L}^*}$  with  $Q^{b_{\mathcal{L}^*}}(\pi_{\mathcal{L}^*}) = Q^b(\pi)$ , and vice versa.*

#### 5.4 $\varepsilon$ -LANDMARKS & Incremental BFBB

As we discussed earlier, if the value of the initial state is not zero, then the empty plan has some positive value, and thus the  $\varepsilon$ -compilation  $\Pi_\varepsilon$  of  $\Pi$  as in Definition 5 will have no landmarks with positive cost. In passing we noted that this small problem can be remedied by considering as “valuable” only facts  $v$  such that both  $u_v(d) > 0$  and  $\langle v/d \rangle \notin s_0$ . We now consider this aspect of OSP more closely, and show how  $\varepsilon$ -landmarks discovery and incremental revelation of plans by BFBB can be combined in a mutually stratifying way.

Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be the OSP task of our interest, and suppose we are *given* a set of plans  $\pi_1, \dots, \pi_n$  for  $\Pi$ . If so, then we are no longer interested in searching for plans that “achieve something,” but in searching for plans that achieve *something beyond what*  $\pi_1, \dots, \pi_n$  *already achieve*. Specifically, let  $s_i = s_0 \llbracket \pi_i \rrbracket$  be the end-state of  $\pi_i$ , and for any set of propositions  $s \subseteq \mathcal{D}$ , let  $\text{goods}(s) \subseteq s$  be the set of all propositions  $\langle v/d \rangle \in s$  such that  $u_v(d) > 0$ . If a new plan  $\pi$  with end-state  $s$  achieves something beyond what  $\pi_1, \dots, \pi_n$  already achieve, then, for *all*  $1 \leq i \leq n$ ,

$$\text{goods}(s) \setminus \text{goods}(s_i) \neq \emptyset.$$

We now put this observation to work.

**Definition 8** *Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$  and a set of reference states  $S_{\text{ref}} = \{s_1, \dots, s_n\}$  of  $\Pi$ , the  $(\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$  is a classical planning task  $\Pi_{(\varepsilon, S_{\text{ref}})} = \langle V_\varepsilon, s_{0\varepsilon}, G_\varepsilon; O_\varepsilon, c_\varepsilon \rangle$  with*

$$\begin{aligned} V_\varepsilon &= V \cup \{x_1, \dots, x_n, \text{search}, \text{collect}\}, \\ &\text{with } \text{dom}(x_i) = \text{dom}(\text{search}) = \text{dom}(\text{collect}) = \{0, 1\}, \\ s_{0\varepsilon} &= s_0 \cup \{\langle \text{search}/1 \rangle, \langle \text{collect}/0 \rangle, \langle x_1/0 \rangle, \dots, \langle x_n/0 \rangle\}, \\ G_\varepsilon &= \{\langle x_1/1 \rangle, \dots, \langle x_n/1 \rangle\}, \\ O_\varepsilon &= \overline{O} \cup \bigcup_{i=1}^n O_i \cup \{\text{finish}\}, \end{aligned}$$

where

$$\bullet \overline{O} = \{\bar{o} \mid o \in O\},$$

$$\text{pre}(\bar{o}) = \text{pre}(o) \cup \{\langle \text{search}/1 \rangle\},$$

$$\text{eff}(\bar{o}) = \text{eff}(o),$$

$$c_\varepsilon(\bar{o}) = c(o).$$

- $O_i = \{o_{i,g} \mid s_i \in S_{ref}, g \in \text{goods}(\mathcal{D}) \setminus s_i\},$

$$\text{pre}(o_{i,g}) = \{g, \langle \text{collect}/1 \rangle\},$$

$$\text{eff}(o_{i,g}) = \{\langle x_i/1 \rangle\},$$

$$c_\varepsilon(o_{i,g}) = 0.$$

•

$$\text{pre}(\text{finish}) = \emptyset,$$

$$\text{eff}(\text{finish}) = \{\langle \text{collect}/1 \rangle, \langle \text{search}/0 \rangle\},$$

$$c_\varepsilon(\text{finish}) = 0.$$

Note that

- the goal  $G_\varepsilon$  cannot be achieved without applying the *finish* operator,
- the operators  $\bar{o}$  can be applied only before *finish*,
- the subgoal achieving operators  $o_{i,g}$  can be applied only after *finish*.

This way, the first part of any plan for  $\Pi_{(\varepsilon, S_{ref})}$  determines a plan for  $\Pi$ , and the second part “verifies” that the end-state of that plan achieves a subset of value-carrying propositions  $\text{goods}(\mathcal{D})$  that is included in no state from  $S_{ref}$ .<sup>9</sup>

**Theorem 18** *Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $S_{ref} = \{s_1, \dots, s_n\}$  be a subset of  $\Pi$ ’s states, and  $L$  be a landmark for  $\Pi_{(\varepsilon, S_{ref})}$  such that  $L \subseteq \bar{O}$ . For any plan  $\pi$  for  $\Pi$  such that  $\text{goods}(s_0[\pi]) \setminus \text{goods}(s_i) \neq \emptyset$  for all  $s_i \in S_{ref}$ ,  $\pi$  contains an instance of at least one operator from  $L' = \{o \mid \bar{o} \in L\}$ .*

*Proof:* Assume to the contrary that there exists a plan  $\pi = \langle o_1, \dots, o_k \rangle$  for  $\Pi$  such that  $\text{goods}(s_0[\pi]) \setminus \text{goods}(s_i) \neq \emptyset$  for all  $s_i \in S_{ref}$ , and yet  $\pi \cap L' = \emptyset$ . Given that, let  $\{g_1, \dots, g_n\}$  be an arbitrary set of propositions from  $\text{goods}(s_0[\pi]) \setminus \text{goods}(s_1), \dots, \text{goods}(s_0[\pi]) \setminus \text{goods}(s_n)$ , respectively. By the construction of  $\Pi_{(\varepsilon, S_{ref})}$ , it is immediate that

$$\pi_{(\varepsilon, S_{ref})} = \langle \bar{o}_1, \dots, \bar{o}_k, \text{finish}, o_{1,g_1}, \dots, o_{n,g_n} \rangle$$

is a plan for  $\Pi_{(\varepsilon, S_{ref})}$  and, by our assumption about  $\pi$  and  $L'$ , it holds that  $\pi_{(\varepsilon, S_{ref})} \cap L = \emptyset$ . This, however, contradicts that  $L$  is a landmark for  $\Pi_{(\varepsilon, S_{ref})}$ .  $\square$

Theorem 18 allows us to define an iterative version of *BFBB*, *inc-compile-and-BFBB*, depicted in Figure 12. The successive iterations of *inc-compile-and-BFBB* correspond to running the regular *BFBB* on successively more informed  $(\varepsilon, S_{ref})$ -compilations of  $\Pi$ , with the states discovered at iteration  $i$  making the  $(\varepsilon, S_{ref})$ -compilation used at iteration  $i + 1$  more informed.

*inc-compile-and-BFBB* maintains as a pair of global variables a set of reference states  $S_{ref}$  and the best solution so far  $n^*$ . At each iteration of the loop, *inc-BFBB*, a modified

9. This “solve & verify” planning technique appears to be helpful in many planning formalism compilations; see, e.g., (Keyder & Geffner, 2009).

```

inc-compile-and-BFBB ( $\Pi = \langle V, O; s_0, c, u, b \rangle$ )
  initialize global variables:
     $n^* := s_0$            // best solution so far
     $S_{\text{ref}} := \{s_0\}$  // current reference states
  loop:
     $\Pi_{(\varepsilon, S_{\text{ref}})} = (\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ 
     $\mathcal{L} :=$  a set of landmarks for  $\Pi_{(\varepsilon, S_{\text{ref}})}$ 
     $lcost :=$  admissible landmark cost function from  $\mathcal{L}$ 
     $\Pi_{\mathcal{L}^*} :=$  budget reducing compilation of  $(\mathcal{L}, lcost)$  into  $\Pi$ 
    if inc-BFBB( $\Pi_{\mathcal{L}^*}, S_{\text{ref}}, n^*$ ) = done:
      return plan for  $\Pi$  associated with  $n^*$ 

inc-BFBB ( $\Pi, S_{\text{ref}}, n^*$ )
  open := new max-heap ordered by  $f(n) = h(s[n], b - g(n))$ 
  open.insert(make-root-node( $s_0$ ))
  closed :=  $\emptyset$  best-cost :=  $\emptyset$ ;
  while not open.empty():
     $n :=$  open.pop-max()
    if goods( $s[n]$ )  $\not\subseteq$  goods( $s'$ ) for all  $s' \in S_{\text{ref}}$ :
       $S_{\text{ref}} := S_{\text{ref}} \cup \{s[n]\}$ 
      if termination criterion: return updated
    if  $f(n) \leq u(s[n^*])$ : break
    // the rest is similar to BFBB in Figure 2
    if  $s[n] \notin$  closed or  $g(n) <$  best-cost( $s[n]$ ):
      closed := closed  $\cup \{s[n]\}$ 
      best-cost( $s[n]$ ) :=  $g(n)$ 
      foreach  $o \in O(s[n])$ :
         $n' :=$  make-node( $s[n][o]$ )
        if  $g(n') > b$  or  $f(n') \leq u(s[n^*])$ : continue
        if  $u(s[n']) > u(s[n^*])$ : update  $n^* := n'$ 
        open.insert( $n'$ )
  return done

```

Figure 12: Iterative *BFBB* with landmark enhancement

version of *BFBB*, is called with an  $(\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ , created on the basis of the *current* pair of  $S_{\text{ref}}$  and  $n^*$ . The reference set  $S_{\text{ref}}$  is then extended by inc-BFBB with all the non-redundant value-carrying states discovered during the search, and  $n^*$  is updated if the search discovers nodes of higher value.

If and when the OPEN list becomes empty or the node  $n$  selected from the list promises less than the lower bound, inc-BFBB returns an indicator, *done*, that the best solution  $n^*$  found so far, across the iterations of inc-compile-and-BFBB, is optimal. In that case, inc-compile-and-BFBB leaves its loop and extracts that optimal plan from  $n^*$ . However, inc-BFBB may also terminate in a different way, if a certain complementary termination

criterion is satisfied. The latter criterion comes to assess whether the updates to  $S_{\text{ref}}$  performed in the current session of *BFBB* warrant updating the  $(\varepsilon, S_{\text{ref}})$ -compilation and restarting the search. If terminated this way, *inc-BFBB* returns a respective indicator, and *inc-compile-and-BFBB* goes into another iteration of its loop, with the *updated*  $S_{\text{ref}}$  and  $n^*$ . We note that, while the optimality of the algorithm holds for *any* such termination condition, the latter should greatly affect the runtime efficiency of the algorithm.

## 5.5 Empirical Evaluation

To evaluate the merits of the landmark-based budget reducing compilation, we have extended our Fast Downward based, prototype OSP solver described in Section 3 with the following components:

- $(\varepsilon, S_{\text{ref}})$ -compilation of OSP tasks  $\Pi$  for arbitrary sets of reference states  $S_{\text{ref}}$ ;
- Generation of disjunctive action landmarks for  $(\varepsilon, S_{\text{ref}})$ -compilations using the LM-Cut procedure (Helmert & Domshlak, 2009) of Fast Downward; and
- The incremental *BFBB* procedure *inc-compile-and-BFBB* as in Figure 12, with the search termination criterion being satisfied (only) if the examined node  $n$  improves over current value lower bound.

After some preliminary evaluation, we also added two optimality preserving enhancements to the search. First, the auxiliary variables of our compilations increase the dimensionality of the problem, and this is well known to negatively affect the quality of the abstraction heuristics (Domshlak et al., 2012). Hence, we devised the projections with respect to the *original* OSP problem  $\Pi$ , and the open list was ordered as if the search is done on the original problem, that is, by

$$h \left( s[n]^{\downarrow V}, b - g(n) + \sum_{v_L \notin s[n]} lcost(L) \right),$$

where  $s^{\downarrow V}$  is the projection of the  $\Pi_{\mathcal{L}^*}$ 's state  $s$  on the variables of the original OSP task  $\Pi$ . This change in heuristic evaluation is sound, as Theorem 17 in particular implies that any admissible heuristic for  $\Pi$  is also an admissible heuristic for  $\Pi_{\mathcal{L}^*}$ , and vice versa. Second, when a new node  $n$  is generated, we check whether

$$g(n) + \sum_{L: \langle v_L/0 \rangle \in s[n]} lcost(L) \geq g(n') + \sum_{L: \langle v_L/0 \rangle \in s[n']} lcost(L),$$

for some previously generated node  $n'$  that corresponds to the same state of the original problem  $\Pi$ , that is,  $s[n']^{\downarrow V} = s[n]^{\downarrow V}$ . If so, then  $n$  is pruned right away. Optimality preservation of this enhancement is established in Lemma 19 and proven in Appendix A, p. 50.

**Lemma 19** *Let  $\Pi$  be an OSP task,  $\Pi_{(\varepsilon, S_{\text{ref}})}$  be a  $(\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ ,  $\mathcal{L}$  be a set of landmarks for  $\Pi_{(\varepsilon, S_{\text{ref}})}$ ,  $lcost$  be an admissible landmark cost function for  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}^*}$  be the*

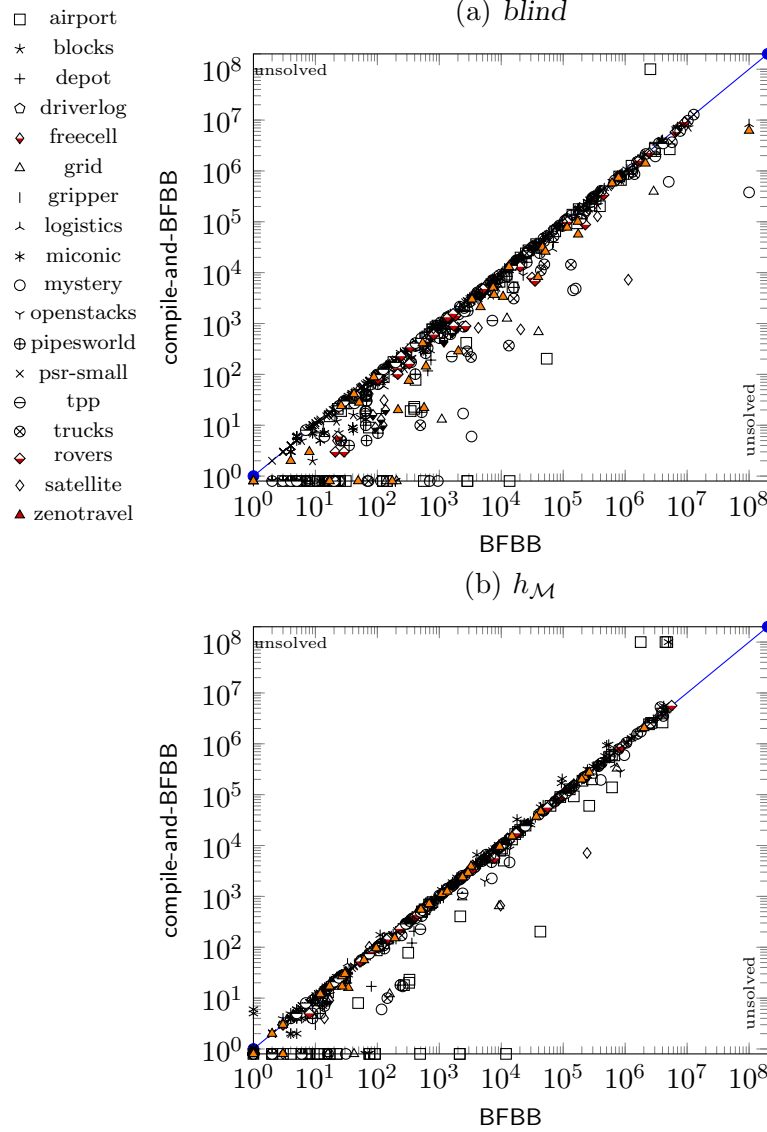


Figure 13: Comparative view of empirical results in terms of expanded nodes, for *BFBB* vs. *compile-and-BFBB*, with (a) blind and (b) abstraction  $h_{\mathcal{M}}$  heuristics

respective budget reducing compilation of  $(\mathcal{L}, lcost)$  into  $\Pi$ . Let  $\pi_1$  and  $\pi_2$  be a pair of plans for  $\Pi_{\mathcal{L}^*}$  with end-states  $s_1$  and  $s_2$ , respectively, such that  $s_1^{\downarrow V} = s_2^{\downarrow V}$  and

$$c_{\mathcal{L}^*}(\pi_1) + \sum_{L: \langle v_L/0 \rangle \in s_1} lcost(L) \geq c_{\mathcal{L}^*}(\pi_2) + \sum_{L: \langle v_L/0 \rangle \in s_2} lcost(L) \quad (20)$$

Then, for any plan  $\pi'_1$  that extends  $\pi_1$ , there exists a plan  $\pi'_2$  that extends  $\pi_2$  such that  $Q^{b_{\mathcal{L}^*}}(\pi'_2) = Q^{b_{\mathcal{L}^*}}(\pi'_1)$ .

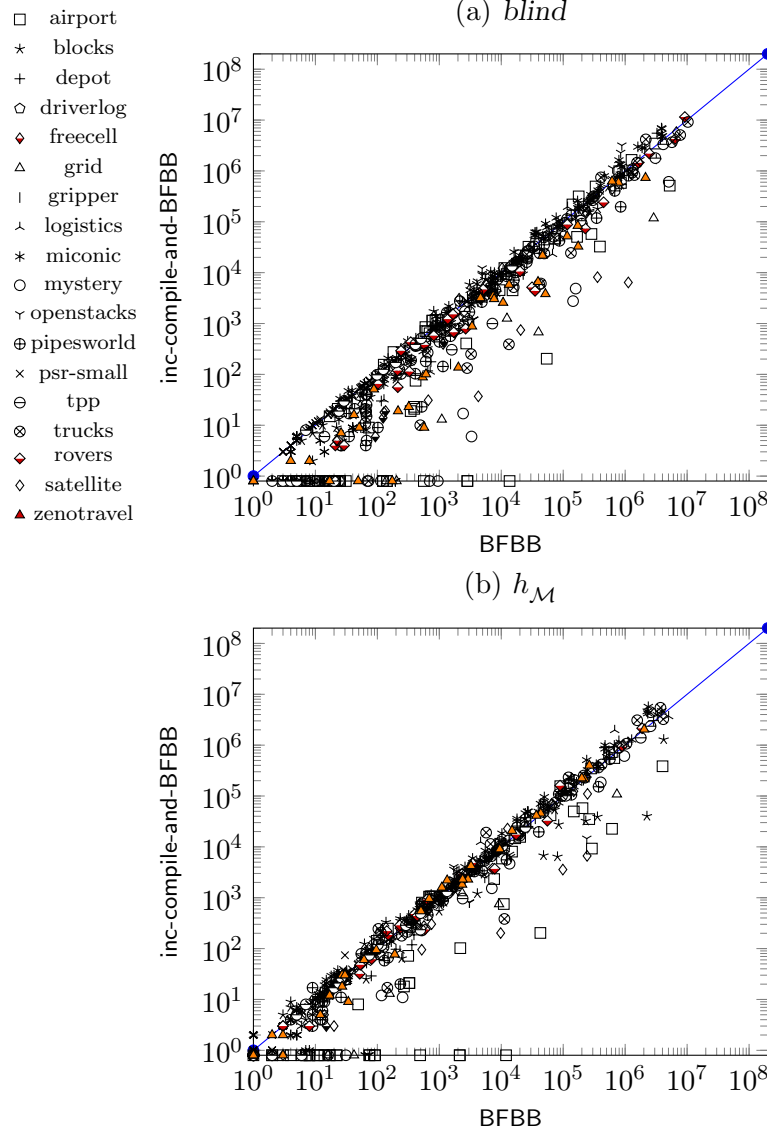


Figure 14: Comparative view of empirical results in terms of expanded nodes, for  $BFBB$  vs.  $inc\text{-}compile\text{-}and\text{-}BFBB$ , with (a) blind and (b) abstraction  $h_{\mathcal{M}}$  heuristics

Our evaluation included the regular  $BFBB$  planning for  $\Pi$ , solving  $\Pi$  using landmark-based compilation via  $compile\text{-}and\text{-}BFBB$ , and the straightforward setting of  $inc\text{-}compile\text{-}and\text{-}BFBB$  described above. All three approaches were evaluated under the blind heuristic and the additive abstraction heuristic  $h_{\mathcal{M}}$  described in Section 3. Figure 7 depicts the results of our evaluation in terms of expanded nodes. Similarly to the experiment reported in Section 3, each task was approached under four different budgets, corresponding to 25%, 50%, 75%, and 100% of the minimal cost needed to achieve all the goals in the task, and each run was restricted to 10 minutes. Figures 13a and 13b compare the performance of  $BFBB$  and

**compile-and-BFBB** in terms of expanded nodes across the four levels of cost budget, under blind (a) and abstraction  $h_{\mathcal{M}}$  (b) heuristics. Figures 14a and 14b provide a similar comparison between *BFBB* and *inc-compile-and-BFBB*.<sup>10</sup> Figures 19-22 and Figures 23-26 in Appendix B provide a more detailed view on the results in Figure 13 and Figure 14, respectively, by breaking them along different levels of cost budget.

As Figure 7 shows, the results were very satisfactory. With no informative heuristic guidance at all, the number of nodes expanded by **compile-and-BFBB** was typically much lower than the number of nodes expanded by *BFBB*, with the difference reaching three orders of magnitude more than once. Of the 760 task/budget pairs behind Figure 7a, 81 pairs were solved by **compile-and-BFBB** with no search at all (by proving that no plan can achieve value higher than that of the initial state), while, unsurprisingly, only 4 of these tasks were solved with no search by *BFBB*.

As expected, the value of landmark-based budget reduction is lower when the search is equipped with a meaningful heuristic (Figure 13b). Yet, even with our abstraction heuristic in hand, the number of nodes expanded by **compile-and-BFBB** was often substantially lower than the number of nodes expanded by *BFBB*. Here, *BFBB* and **compile-and-BFBB** solved with no search 39 and 85 task/budget pairs, respectively. Finally, despite the rather ad hoc setting of our incremental *inc-compile-and-BFBB* procedure, switching from **compile-and-BFBB** to *inc-compile-and-BFBB* was typically beneficial. Obviously, much deeper investigation and development of *inc-compile-and-BFBB* is still required, especially around the flexibility with respect to the iteration termination criterion.

## 6. Summary and Future Work

Deterministic oversubscription planning captures the computational core of one of most practically important setups of automated action selection, and yet, despite the apparent interest in this problem, it has not been sufficiently investigated. In this work, we stepped towards translating the spectacular advances in classical deterministic planning to deterministic OSP. Tracing the key sources of progress in classical planning, we identified a severe lack of effective approximations for OSP, and worked towards bridging this gap.

Our focus was on two classes of approximation techniques that underly most state-of-the-art optimal heuristic-search solvers for classical planning, namely state-space abstractions and goal-reachability landmarks. First, we defined the notion of additive abstractions for OSP, studied the complexity of deriving effective abstractions from a rich space of hypotheses, and reveal some substantial, empirically relevant islands of tractability of this abstraction discovery problem. Next, we showed how standard goal-reachability landmarks of certain classical planning tasks can be compiled into the OSP task of interest, resulting in an equivalent OSP task with a lower cost allowance, and thus with a, sometimes dramatically, smaller search space.

All the techniques proposed here satisfy the properties required by the efficient search algorithms for optimal OSP. However, we believe that these techniques, and especially landmark-based budget reducing compilations, should be as beneficial in satisficing OSP as

---

10. We do not compare here between the running times, but the per-node CPU time overhead due to landmark-based budget reduction was  $\leq 10\%$ .



in optimal OSP, and this especially because the difference between optimal and satisficing planning appears to be much smaller in OSP than in classical deterministic planning.

Many interesting questions remain open for future work, and in general, the prospects for further developments in oversubscription planning appear now quite promising. Within the specific context of our work, the two most interesting questions for us are optimization of value partitions given cost partition, that is, optimizing abstraction discovery in  $\mathbf{A}_p(\mathbf{c}, -, -)$ , and a thorough investigation of the interleaved landmark discovery and search for OSP that was introduced in Section 5.4. Among the prospective research directions on a broader perspective we would like to emphasize the following candidates for investigation.

- Following the work of Katz and Domshlak (2010a) on *implicit abstractions* for classical planning, it is only natural to investigate the computational merits of implicit abstractions for OSP. In particular, this thread of investigation will unavoidably push towards a better understanding of the computational tractability boundaries of deterministic OSP.
- The basic model of deterministic planning in Section 2.1 was used to provide a unifying comparative view on the basic models of classical, cost-bounded, net-benefit, and oversubscription planning, but not beyond that. One practically motivated extension of this model is to lift action costs to vectors of action costs. Such a variant of cost-bounded planning is already investigated (Nakhost et al., 2012), and it is only natural to examine this extension in the context of OSP. Note that, at the level of the model, adding cost measures shifts problem solving from shortest path(s) to restricted shortest path(s) problems. While the latter is already NP-hard (Handler & Zang, 1980), similarly to the Knapsack problem, it can be solved in pseudo-polynomial time (Desrochers & Soumis, 1988).

## Acknowledgments

This work was partially supported by the EOARD grant FA8655-12-1-2096, and the ISF grant 1045/12.

## Appendix A. Proofs

**Theorem 2** *Given an OSP task  $\Pi = \langle V, s_0, u; O, c, b \rangle$  and a homomorphic abstraction skeleton  $\mathcal{AS} = \{(G_1, \alpha_1), \dots, (G_k, \alpha_k)\}$  of  $M_\Pi$ ,*

- (1) *for each cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b}^* \in \mathbf{B}_p$  such that  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b}^*) \in_s \mathcal{AS}$  for all value partitions  $\mathbf{u} \in \mathbf{U}_p$ .*
- (2) *for each budget partition  $\mathbf{b} \in \mathbf{B}_p$ , there exists a cost partition  $\mathbf{c}^* \in \mathbf{C}_p$  such that  $\mathcal{M}(\mathbf{c}^*, \mathbf{u}, \mathbf{b}) \in_s \mathcal{AS}$  for all value partitions  $\mathbf{u} \in \mathbf{U}_p$ .*

*Proof:* Let  $\pi = \langle (s_0, o_1, s_1), (s_1, o_2, s_2), \dots, (s_{n-1}, o_n, s_n) \rangle$  be an optimal  $s_0$ -plan for  $M_\Pi$ , and, for  $i \in [k]$ , let  $\pi_i = \langle (\alpha_i(s_0), o_1, \alpha_i(s_1)), \dots, (\alpha_i(s_{n-1}), o_n, \alpha_i(s_n)) \rangle$  be the mapping of  $\pi$  to  $G_i$ . Since  $\mathcal{AS}$  is homomorphic, the paths  $\pi_1, \dots, \pi_k$  are well-defined.

- (1) Given a cost partition  $\mathbf{c} \in \mathbf{C}_p$ , let budget profile  $\mathbf{b}^* \in \mathbf{B}$  be defined as  $\mathbf{b}^*[i] = \sum_{j \in [n]} \mathbf{c}[i](a_j)$ , for  $i \in [k]$ . First, note that  $\mathbf{b}^* \in \mathbf{B}_p$  since

$$\sum_{i \in [k]} \mathbf{b}^*[i] = \sum_{i \in [k]} \sum_{j \in [n]} \mathbf{c}[i](o_j) \stackrel{(\dagger)}{\leq} \sum_{j \in [n]} c(o_j) \stackrel{(\ddagger)}{\leq} b,$$

where  $(\dagger)$  is by  $\mathbf{c}$  being a cost partition, and  $(\ddagger)$  is by  $\pi$  being an  $s_0$ -plan for  $M_\Pi$ . Second, for any  $\mathbf{u} \in \mathbf{U}$ , by the construction of  $\mathbf{b}^*$ ,  $\pi_i$  is an  $\alpha_i(s_0)$ -plan for the abstract model  $M_i^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)}$ . Now, let  $\mathbf{u} \in \mathbf{U}_p$ , and for  $i \in [k]$ , let  $\pi_i^*$  be an optimal  $\alpha_i(s_0)$ -plan for  $M_i^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)}$ . We have

$$\sum_{i \in [k]} Q^{\mathbf{b}^*[i]}(\pi_i^*) \stackrel{(\dagger)}{\geq} \sum_{i \in [k]} Q^{\mathbf{b}^*[i]}(\pi_i) \stackrel{(\ddagger)}{\geq} Q^b(\pi), \quad (21)$$

where  $(\dagger)$  is by optimality of  $\pi_i^*$ , and  $(\ddagger)$  is by  $\alpha_i(s_n)$  being the end-state of  $\pi_i$  and  $\mathbf{u}$  being a value partition. Therefore,  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  induces an additive abstraction for  $\Pi$ , that is,  $\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)} \in_{\mathcal{AS}} M_\Pi$ .

- (2) Given a budget partition  $\mathbf{b}$ , let cost profile  $\mathbf{c}^* \in \mathbf{C}$  be defined as  $\mathbf{c}^*[i](o) = c(o) \cdot \frac{\mathbf{b}[i]}{b}$ , for all operators  $o \in O$ , and all  $i \in [k]$ . First, we have  $\mathbf{c}^* \in \mathbf{C}_p$  since  $\mathbf{b} \in \mathbf{B}_p$  implies  $\frac{1}{b} \sum_{i \in [k]} \mathbf{b}[i] \in [0, 1]$ . Second, for any  $\mathbf{u} \in \mathbf{U}$ , by the construction of  $\mathbf{c}^*$ ,  $\pi_i$  is an  $\alpha_i(s_0)$ -plan for  $M_i^{(\mathbf{c}^*, \mathbf{u}, \mathbf{b})}$ . Following now exactly the same line of reasoning as the one around Eq. 21 above accomplishes the proof that  $\mathcal{M}^{(\mathbf{c}^*, \mathbf{u}, \mathbf{b})} \in_{\mathcal{AS}} M_\Pi$  for any  $\mathbf{u} \in \mathbf{U}_p$ .

□

**Lemma 6** *The algorithm in Figure 8a computes  $\kappa(\mathbf{u})$ .*

*Proof:* Due to the boundness and non-emptiness of the polytope induced by  $\mathcal{L}_1(m)$ , the termination of the algorithm is straightforward. Thus, given a strong 0-binary partition  $\mathbf{u}$ , the only question is whether the value with which the algorithm terminates is  $\kappa(\mathbf{u})$ . First, let us show that:

- ( $\dagger$ ) For  $m \in [k]$ , if  $\mathbf{x}$  is a solution of  $\mathcal{L}_1(m)$ , then  $\mathbf{x}[\xi] \leq b$  if and only if, for *each* cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  such that  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$  is an abstraction for  $M_\Pi$  and  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0) \geq m\sigma$ .

( $\Leftarrow$ ) Assume to the contrary that, for *each* cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  with  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0) \geq m\sigma$ , and yet  $\mathbf{x}[\xi] > b$ . Given the values provided by  $\mathbf{x}$  to the cost variables  $\bigcup_{o \in O} \{\mathbf{c}[i](o)\}$ , let  $\mathbf{c}$  be the corresponding cost partition, and  $\delta_1, \dots, \delta_k$  be the induced lengths of the shortest paths from  $\alpha_1(s_0), \dots, \alpha_k(s_0)$  to  $\sigma$ -valued states in  $G_1, \dots, G_k$ , respectively. By our assumption, let  $\mathbf{b}$  be a budget partition such that  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0) \geq m\sigma$ . First, by the definition of strong 0-binary value partitions,  $h_{\mathcal{M}^{(\mathbf{c}, \mathbf{u}, \mathbf{b})}}(s_0) \geq m\sigma$  implies that there exists  $Z \subseteq [k]$ ,  $|Z| = m$  such that, for  $i \in Z$ ,  $\mathbf{b}[i] \geq \delta_i$ .

Second, constraint (10c), maximization of  $\xi$ , and the fact that the only bound on each  $\mathbb{b}[i]$  is by  $\delta_i$  imply together that, for  $i \in Z$ ,  $\mathbf{x}[\mathbb{b}[i]] = \delta_i$ . Putting things together, we obtain

$$b \geq \sum_{i \in Z}^{\mathbf{b} \in \mathbf{B}_p} \mathbf{b}[i] \geq \sum_{i \in Z} \delta_i = \sum_{i \in Z} \mathbf{x}[\mathbb{b}[i]] \stackrel{(10c)}{\geq} \xi,$$

contradicting our assumption.

( $\Rightarrow$ ) Assume to the contrary that,  $\mathbf{x}[\xi] \leq b$ , and yet there exists a cost partition  $\mathbf{c} \in \mathbf{C}_p$  such that, for all budget partitions  $\mathbf{b} \in \mathbf{B}_p$  with  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$ , we have  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) < m\sigma$ .

Let the shortest path lengths  $\delta_1, \dots, \delta_k$  be defined as above, but now with respect to the specific cost partition  $\mathbf{c}$  from the assumption. Likewise, let  $\mathbf{x}_{\mathbf{c}}$  be a solution to  $\mathcal{L}_1(m)$  with an extra constraint on the cost variables  $\bigcup_{o \in O} \{\mathbf{c}[i](o)\}$  to be assigned to  $\mathbf{c}$ . Since the objective in  $\mathcal{L}_1(m)$  is to maximize the value of  $\xi$ , we have

$$\mathbf{x}[\xi] \geq \mathbf{x}_{\mathbf{c}}[\xi]. \quad (22)$$

Now, let

$$Z = \operatorname{argmax}_{Z' \subseteq [k], |Z'|=m} \sum_{i \in Z'} \delta_i.$$

Together, constraint (10c), maximization of  $\xi$ , and the fact that the only bound on each  $\mathbb{b}[i]$  is by  $\delta_i$  (via the cost variables) imply that

$$\mathbf{x}_{\mathbf{c}}[\xi] = \sum_{i \in Z} \mathbf{x}_{\mathbf{c}}[\mathbb{b}[i]] = \sum_{i \in Z} \delta_i. \quad (23)$$

In turn, together with  $\mathbf{x}[\xi] \leq b$  and Eq. 22, Eq. 23 implies that

$$\mathbf{b}[i] = \begin{cases} \mathbf{x}_{\mathbf{c}}[\mathbb{b}[i]], & i \in Z \\ 0, & \text{otherwise} \end{cases},$$

is a budget partition with  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$ , and  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq m\sigma$ , contradicting our assumption.

Having proved the sub-claim ( $\dagger$ ), which basically captures the semantics of  $\mathcal{L}_1(m)$ , suppose that the algorithm terminates within the loop, and returns  $m\sigma$  for some  $m > 0$ . By the construction of the algorithm, if  $\mathbf{x}$  is a solution of  $\mathcal{L}_1(m)$ , then  $\mathbf{x}[\xi] \leq b$ . By ( $\dagger$ ), for each cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$  such that  $h_{(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s) \geq m\sigma$ . If  $m = k$ , then trivially  $\kappa_s(\mathbf{u}) = m\sigma$ . Otherwise, if  $m < k$ , we know that the algorithm did not terminate at the previous iteration corresponding to  $m + 1$ . Again, ( $\dagger$ ) then implies that there exists a cost partition  $\mathbf{c} \in \mathbf{C}_p$  for which no  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$  will induce  $h_{(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s) \geq (m + 1)\sigma$ . Hence, by the definition of  $\kappa_s(\mathbf{u})$ ,  $\kappa_s(\mathbf{u}) < (m + 1)\sigma$ , and in turn, since  $\mathbf{u}$  is a strong 0-binary value partition, that implies  $\kappa_s(\mathbf{u}) = m\sigma$ . Finally, if the algorithm terminates after the loop and returns 0, then precisely the same argument on the basis of ( $\dagger$ ) implies  $\kappa_s(\mathbf{u}) = 0$ .  $\square$

**Lemma 9** *For any  $0 < \epsilon < \min_{i \in [k]} \sigma_i$ , the algorithm in Figure 9a computes  $\kappa_\epsilon$  such that  $\kappa_\epsilon - \kappa(\mathbf{u}) \leq \epsilon$ .*

*Proof:* The arguments for the boundness and non-emptiness of the polytope induced by  $\mathcal{L}_2(v)$  are precisely the same as for the polytope of  $\mathcal{L}_1(m)$  studied in Lemma 6, and thus the termination of the algorithm is straightforward. In what follows, we prove that the value returned by the algorithm satisfies the claim of the lemma. Let  $\mathbf{u}$  be the given 0-binary partition. Similarly to the proof of Lemma 9, first we prove a sub-claim that:

- (†) For  $v \in \mathbb{R}^{0+}$ , if  $\mathbf{x}$  is a solution of  $\mathcal{L}_2(v)$ , then  $\mathbf{x}[\xi] \leq b$  if and only if, for *each* cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  such that  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$  is an abstraction for  $M_\Pi$  and  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq v$ .

The proof of (†) mirrors the proof of the respective sub-claim in Lemma 5, *mutatis mutandis*, and thus it is provided here only for ease of verification.

( $\Leftarrow$ ) Assume to the contrary that, for *each* cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  with  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq v$ , and yet  $\mathbf{x}[\xi] > b$ .

Given the values provided by  $\mathbf{x}$  to the cost variables  $\bigcup_{o \in O} \{\mathbf{c}[i](o)\}$ , let  $\mathbf{c}$  be the corresponding cost partition, and, for  $i \in [k]$ , let  $\delta_i$  be the induced length of the shortest path from  $\alpha_i(s_0)$  to the  $\sigma_i$ -valued states in  $G_i$ . By our assumption, let  $\mathbf{b}$  be a budget partition such that  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq v$ . First, by the definition of 0-binary value partitions,  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq v$  implies that there exists  $Z \subseteq [k]$ ,  $\sum_{i \in Z} \sigma_i \geq v$  such that, for  $i \in Z$ ,  $\mathbf{b}[i] \geq \delta_i$ . Second, constraint (11c), maximization of  $\xi$ , and the fact that the only bound on each  $\mathbf{b}[i]$  is by  $\delta_i$ , imply together that, for  $i \in Z$ ,  $\mathbf{x}[\mathbf{b}[i]] = \delta_i$ . Putting things together, we obtain

$$b \geq \sum_{i \in Z}^{\mathbf{b} \in \mathbf{B}_p} \mathbf{b}[i] \geq \sum_{i \in Z} \delta_i = \sum_{i \in Z} \mathbf{x}[\mathbf{b}[i]] \stackrel{(11c)}{\geq} \xi,$$

contradicting our assumption.

( $\Rightarrow$ ) Assume to the contrary that,  $\mathbf{x}[\xi] \leq b$ , and yet there exists a cost partition  $\mathbf{c} \in \mathbf{C}_p$  such that, for all budget partitions  $\mathbf{b} \in \mathbf{B}_p$  with  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$ , we have  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) < v$ .

Let the shortest path lengths  $\delta_1, \dots, \delta_k$  be defined as above, but now with respect to the specific cost partition  $\mathbf{c}$  from the assumption. Likewise, let  $\mathbf{x}_\mathbf{c}$  be a solution to  $\mathcal{L}_2(v)$  with an extra constraint on the cost variables  $\bigcup_{o \in O} \{\mathbf{c}[i](o)\}$  to be assigned to  $\mathbf{c}$ . Since the objective in  $\mathcal{L}_2(v)$  is to maximize the value of  $\xi$ , we have

$$\mathbf{x}[\xi] \geq \mathbf{x}_\mathbf{c}[\xi]. \quad (24)$$

Now, let

$$Z = \underset{\substack{Z' \subseteq [k], \\ \sum_{i \in Z'} \sigma_i \geq v}}{\operatorname{argmax}} \sum_{i \in Z'} \delta_i.$$

Together, constraint (11c), maximization of  $\xi$ , and the fact that the only bound on each  $\mathbf{b}[i]$  is by  $\delta_i$  (via the cost variables) imply that

$$\mathbf{x}_\mathbf{c}[\xi] = \sum_{i \in Z} \mathbf{x}_\mathbf{c}[\mathbf{b}[i]] = \sum_{i \in Z} \delta_i. \quad (25)$$

In turn, together with  $\mathbf{x}[\xi] \leq b$  and Eq. 24, Eq. 25 implies that

$$\mathbf{b}[i] = \begin{cases} \mathbf{x}_\mathbf{c}[\mathbf{b}[i]], & i \in Z \\ 0, & \text{otherwise} \end{cases},$$

is a budget partition with  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{A}_p$ , and  $h_{\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s_0) \geq v$ , contradicting our assumption.

This finalizes the proof of the sub-claim  $(\dagger)$ . Now, consider the interval end-points  $\alpha$  and  $\beta$  at the termination of the while-loop. If  $\beta = \sum_{i \in [k]} \sigma_i$ , then trivially  $\kappa(\mathbf{u}) \leq \beta$ . Otherwise, if  $\beta < \sum_{i \in [k]} \sigma_i$ , then, by the construction of the algorithm, at some iteration of the while loop, a test `always-achievable`( $\beta$ ) was issued, came negative, and thus, for the solutions  $\mathbf{x}_\beta$  of  $\mathcal{L}_2(\beta)$ , we have  $\mathbf{x}_\beta[\xi] > b$ . Hence, by  $(\dagger)$ ,  $\kappa(\mathbf{u}) < \beta$ . Now, if  $\alpha \neq 0$ , then, by the construction of the algorithm, at some iteration of the while loop, a test `always-achievable`( $\alpha$ ) was issued, came positive, and thus, for the solutions  $\mathbf{x}_\alpha$  of  $\mathcal{L}_2(\alpha)$ , we have  $\mathbf{x}_\alpha[\xi] \leq b$ . Hence, by  $(\dagger)$ ,  $\kappa(\mathbf{u}) \geq \alpha$ . Putting these properties on  $\alpha$  and  $\beta$  together with the while-loop's termination condition  $\beta - \alpha \leq \epsilon$  implies  $\kappa_\epsilon - \kappa(\mathbf{u}) = \beta - \kappa(\mathbf{u}) \leq \epsilon$ . Finally, if  $\alpha = 0$ , then  $\epsilon < \min_{i \in [k]} \sigma_i$  implies  $\beta < \min_{i \in [k]} \sigma_i$ . In turn, since  $\kappa(\mathbf{u})$  corresponds to a sum of values of some states in the  $k$  models of  $\mathcal{M}(\mathbf{c}, \mathbf{u}, \mathbf{b})$ ,  $\kappa(\mathbf{u}) \leq \beta$  concluded above implies  $\kappa_\epsilon = \kappa(\mathbf{u}) = 0$ .  $\square$

**Theorem 16** *Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ ,  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}}$  be the respective budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $Q^b(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}}$  for  $\Pi_{\mathcal{L}}$  with  $Q^{b_{\mathcal{L}}}(\pi_{\mathcal{L}}) = Q^b(\pi)$ , and vice versa.*

*Proof:* Let  $\pi_{\mathcal{L}}$  be a plan for  $\Pi_{\mathcal{L}}$ , and let  $\pi$  be the operator sequence obtained by replacing all operators  $\bar{o}$  from  $\bigcup_{i=1}^n O_{L_i}$  along  $\pi_{\mathcal{L}}$  with the respective operators  $o \in O$ . By the definition of the action set of  $\Pi_{\mathcal{L}}$  in Eq. 15, we have  $\pi$  applicable in  $s_0$ , and  $s_0[\llbracket \pi \rrbracket] = s_{0_{\mathcal{L}}}[\llbracket \pi_{\mathcal{L}} \rrbracket] \setminus \bigcup_{i=1}^n \text{dom}(v_{L_i})$ . Thus,  $Q^b(\pi) = Q^{b_{\mathcal{L}}}(\pi_{\mathcal{L}})$ . Likewise, by, again, the definition of the action set of  $\Pi_{\mathcal{L}}$  in Eq. 15 and the fact that no operator in  $O_{\mathcal{L}}$  achieves the control propositions  $\{\langle v_{L_1}/1 \rangle, \dots, \langle v_{L_n}/1 \rangle\}$ , we have  $|O_{L_i} \cap \pi_{\mathcal{L}}| \leq 1$ . From that, we have

$$c(\pi) \leq c_{\mathcal{L}}(\pi_{\mathcal{L}}) + \sum_{i=1}^n lcost(L_i).$$

In turn,  $b = b_{\mathcal{L}} + \sum_{i=1}^n lcost(L_i)$  by Eq. 14, and  $c_{\mathcal{L}}(\pi_{\mathcal{L}}) \leq b_{\mathcal{L}}$  by the virtue of  $\pi_{\mathcal{L}}$  being a plan for  $\Pi_{\mathcal{L}}$ . Therefore, it holds that  $c(\pi) \leq b$ , and thus  $\pi$  is a plan for  $\Pi$ .

In the opposite direction, let  $\pi$  be a plan for  $\Pi$  with  $Q^b(\pi) > 0$ , and let  $\pi_{\mathcal{L}}$  be an operator sequence obtained by replacing, for each  $\varepsilon$ -landmark  $L \in \mathcal{L}$ , every first occurrence of an operator from  $L$  with the respective “cost reduced” operator from  $O_L$ . It is easy to verify that  $\pi_{\mathcal{L}}$  is applicable in  $s_{0_{\mathcal{L}}}$ , and that  $Q^{b_{\mathcal{L}}}(\pi_{\mathcal{L}}) = Q^b(\pi)$ . Likewise, by the definition of  $\varepsilon$ -landmarks, every  $L \in \mathcal{L}$  will have a presence along  $\pi$ . From that, we have

$$c(\pi_{\mathcal{L}}) = c(\pi) - \sum_{i=1}^n lcost(L_i) \leq b - \sum_{i=1}^n lcost(L_i) = b_{\mathcal{L}},$$

where the first equality is by pairwise disjointness of  $\{L_1, \dots, L_n\}$ , the inequality is by  $\pi$  being a plan for  $\Pi$ , and the second equality is by Eq. 14. Thus,  $\pi_{\mathcal{L}}$  is a plan for  $\Pi_{\mathcal{L}}$ .  $\square$

**Theorem 17** *Let  $\Pi = \langle V, s_0, u; O, c, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of  $\varepsilon$ -landmarks for  $\Pi$ ,  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}^*}$  be the (generalized) budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $Q^b(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}^*}$  for  $\Pi_{\mathcal{L}^*}$  with  $Q^{b_{\mathcal{L}^*}}(\pi_{\mathcal{L}^*}) = Q^b(\pi)$ , and vice versa.*

*Proof:* Let  $\pi_{\mathcal{L}^*}$  be a plan for  $\Pi_{\mathcal{L}^*}$ , and let  $\pi$  be the operator sequence obtained by (i) replacing all operators  $\bar{o}$  with the respective operators  $o \in O$ , and (ii) removal of all *get* operators. By Eq. 17, we have  $\pi$  applicable in  $s_0$ , and  $s_0[\pi] = s_{0\mathcal{L}^*}[\pi_{\mathcal{L}^*}] \setminus \{\langle v_{L_1}/1 \rangle, \dots, \langle v_{L_n}/1 \rangle\}$ . Thus,  $Q^b(\pi) = Q^{b_{\mathcal{L}^*}}(\pi_{\mathcal{L}^*})$ . Now, for each  $\varepsilon$ -landmark  $L \in \mathcal{L}$ , let  $\xi(L)$  be the number of instances of the cost reduced counterparts  $\bar{o}$  of the operators from  $L$  along  $\pi_{\mathcal{L}^*}$ . By Eqs. 17 and 18, for each  $L \in \mathcal{L}$ ,  $\pi_{\mathcal{L}^*}$  must contain at least  $\xi(L) - 1$  instances of operator  $get(L)$ . From that, we have

$$\begin{aligned}
c(\pi) &\leq c_{\mathcal{L}^*}(\pi_{\mathcal{L}^*}) + \sum_{\bar{o} \in \pi_{\mathcal{L}^*}} \sum_{L \in \mathcal{L}(o)} lcost(L) - \sum_{L \in \mathcal{L}} (\xi(L) - 1) \cdot lcost(L) \\
&= c_{\mathcal{L}^*}(\pi_{\mathcal{L}^*}) + \sum_{L \in \mathcal{L}} \xi(L) \cdot lcost(L) - \sum_{L \in \mathcal{L}} (\xi(L) - 1) \cdot lcost(L) \\
&= c_{\mathcal{L}^*}(\pi_{\mathcal{L}^*}) + \sum_{L \in \mathcal{L}} lcost(L) \\
&\leq b_{\mathcal{L}^*} + \sum_{L \in \mathcal{L}} lcost(L) \\
&= b,
\end{aligned}$$

and thus  $\pi$  is a plan for  $\Pi$ .

In the opposite direction, let  $\pi = \langle o_1, \dots, o_m \rangle$  be a plan for  $\Pi$  with  $Q^b(\pi) > 0$ . By the definition of  $\varepsilon$ -landmarks, every  $L \in \mathcal{L}$  will have a presence along  $\pi$ , and let  $o_{f(i)}$ ,  $1 \leq f(i) \leq n$ , be the first occurrence of an operator from  $L_i$  along  $\pi$ . Since  $\varepsilon$ -landmarks in  $\mathcal{L}$  are not necessarily disjoint, we may have  $f(i) = f(j)$  for some pairs  $1 \leq i \neq j \leq n$ . Let  $\rho = \langle o_{(1)}, \dots, o_{(k)} \rangle$ ,  $k \leq n$ , be a sequence of operators obtained from  $\{o_{f(1)}, \dots, o_{f(n)}\}$  by (a) duplicate elimination, and (b) ordering consistently with  $\pi$ . Let  $\pi_{\mathcal{L}^*}$  be an operator sequence obtained from  $\pi$  based on  $\rho$  by

- (1) replacing each  $o_{(i)}$  with  $\bar{o}_{(i)}$ , and
- (2) inserting right before each  $\bar{o}_{(i)}$  an arbitrary ordered sequence of actions

$$\bigcup_{j=1}^{i-1} \{get(L) \mid L \in \mathcal{L}, \{o_{(j)}, o_{(i)}\} \subseteq L\}.$$

It is not hard to verify that  $\pi_{\mathcal{L}^*}$  is applicable in  $s_{0\mathcal{L}^*}$ , and that  $Q^{b_{\mathcal{L}^*}}(\pi_{\mathcal{L}^*}) = Q^b(\pi)$ . Now, step (1) of expanding  $\pi$  to  $\pi_{\mathcal{L}^*}$  (over all  $1 \leq i \leq k$ ) reduces the cost of the operator sequence by

$$\sum_{i=1}^k \sum_{L \in \mathcal{L}(o_{(i)})} lcost(L) = \sum_{L \in \mathcal{L}} \mu(L) \cdot lcost(L),$$

where  $\mu(L) = |L \cap \rho|$ . Step (2) of expanding  $\pi$  to  $\pi_{\mathcal{L}^*}$  increases the cost of the operator sequence by  $\sum_{L \in \mathcal{L}} (\mu(L) - 1) \cdot lcost(L)$ . Thus,

$$c_{\mathcal{L}^*}(\pi_{\mathcal{L}^*}) = c(\pi) - \sum_{L \in \mathcal{L}} lcost(L) \leq b - \sum_{L \in \mathcal{L}} lcost(L) = b_{\mathcal{L}^*},$$

that is,  $\pi_{\mathcal{L}^*}$  is a plan for  $\Pi_{\mathcal{L}^*}$ .  $\square$

**Lemma 19** *Let  $\Pi$  be an OSP task,  $\Pi_{(\varepsilon, S_{\text{ref}})}$  be a  $(\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ ,  $\mathcal{L}$  be a set of landmarks for  $\Pi_{(\varepsilon, S_{\text{ref}})}$ ,  $lcost$  be an admissible landmark cost function for  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}^*}$  be the respective budget reducing compilation of  $(\mathcal{L}, lcost)$  into  $\Pi$ . Let  $\pi_1$  and  $\pi_2$  be a pair of plans for  $\Pi_{\mathcal{L}^*}$  with end-states  $s_1$  and  $s_2$ , respectively, such that  $s_1^{\downarrow V} = s_2^{\downarrow V}$  and*

$$c_{\mathcal{L}^*}(\pi_1) + \sum_{L: \langle v_L/0 \rangle \in s_1} lcost(L) \geq c_{\mathcal{L}^*}(\pi_2) + \sum_{L: \langle v_L/0 \rangle \in s_2} lcost(L) \quad (20)$$

*Then, for any plan  $\pi'_1$  that extends  $\pi_1$ , there exists a plan  $\pi'_2$  that extends  $\pi_2$  such that  $Q^{b_{\mathcal{L}^*}}(\pi'_2) = Q^{b_{\mathcal{L}^*}}(\pi'_1)$ .*

*Proof:* Under the notation in the claim, the proof is by a constructive mapping of the plan  $\pi'_1$  to the corresponding plan  $\pi'_2$ .

First, we derive from  $\pi'_1$  a plan  $\rho'_1$  for  $\Pi$  by (i) removing the *finish* operator and all the *get*( $\cdot$ ) operators, and (ii) replacing all instances of each discounted operator  $\bar{o}$  with instances of the respective original operator  $o$ . This results in a plan  $\rho'_1 := \rho_1 \cdot \rho_{1e}$  for  $\Pi$  with  $s_0[\rho_1] = s_1^{\downarrow V}$  and  $c(\rho_1) = c_{\mathcal{L}^*}(\pi_1) + \sum_{L: \langle v_L/0 \rangle \in s_1} lcost(L)$ . To see the latter, for each operator  $\omega \in O_{\mathcal{L}^*}$ , let  $\kappa(\omega) \geq 0$  denote the number of instances of  $\omega$  along  $\pi_1$ . Given that, we have

$$\begin{aligned} c(\rho_1) &= c_{\mathcal{L}^*}(\pi_1) - \sum_{L \in \mathcal{L}} \kappa(\text{get}(L)) lcost(L) + \sum_{L \in \mathcal{L}} lcost(L) \sum_{o: L \in \mathcal{L}(o)} \kappa(\bar{o}) \\ &= c_{\mathcal{L}^*}(\pi_1) + \sum_{L \in \mathcal{L}} lcost(L) \left[ \left( \sum_{o: L \in \mathcal{L}(o)} \kappa(\bar{o}) \right) - \kappa(\text{get}(L)) \right] \\ &= c_{\mathcal{L}^*}(\pi_1) + \sum_{L \in \mathcal{L}} lcost(L) \cdot \mathbb{1}_{\langle v_L/0 \rangle \in s_1} \\ &= c_{\mathcal{L}^*}(\pi_1) + \sum_{L: \langle v_L/0 \rangle \in s_1} lcost(L), \end{aligned} \quad (26)$$

where the second and fourth equalities are just formula manipulations, the first equality is direct from the construction of  $\rho_1$ , and the third equality is by the definition of the budget reducing compilation, and specifically, by Eqs. 17 and 18.

Similarly to the construction of  $\rho_1$  from  $\pi_1$ , we can construct  $\rho_2$  from  $\pi_2$ , with and  $s_0[\rho_2] = s_2^{\downarrow V}$  and  $c(\rho_2) = c_{\mathcal{L}^*}(\pi_2) + \sum_{L: \langle v_L/0 \rangle \in s_2} lcost(L)$ . Thus, by Eq. 20,  $c(\rho_1) \geq c(\rho_2)$ , and also, by the setting of the claim,  $s_0[\rho_1] = s_0[\rho_2]$ . Hence,  $\rho'_2 = \rho_2 \cdot \rho_{1e}$  is also a plan for  $\Pi$ , and  $Q^b(\rho'_1) = Q^b(\rho'_2)$ .

As the last step, we now construct from  $\rho'_2$  a plan  $\pi'_2$  for  $\Pi_{\mathcal{L}^*}$  as in the claim. First, by the properties of  $\pi_2$  in the claim, the plan  $\rho_2$  for  $\Pi$  achieves all the landmarks  $\mathcal{L}_{\notin s_2} = \{L \mid \langle v_L/0 \rangle \in s_2\}$ . Second, by the definition of the landmark set  $\mathcal{L}$ ,  $\rho_{1e}$  must satisfy the rest of the landmarks, that is,  $\mathcal{L}_{\in s_2} = \{L \mid \langle v_L/1 \rangle \in s_2\}$ . Let us denote the operator instances along  $\rho_{1e}$  as  $\langle o_1, \dots, o_k \rangle$ ,  $k = |\rho_{1e}|$ , and let  $\{\mathcal{L}_1, \dots, \mathcal{L}_k\}$  be a partition of  $\mathcal{L}_{\in s_2}$  with  $\mathcal{L}_i \subseteq \mathcal{L}_{\in s_2}$  being the subset of all landmarks from  $\mathcal{L}_{\in s_2}$  for which  $o_i$  is their first achiever along  $\rho_{1e}$ .

Given that, consider an operator sequence  $\pi_{2e} := \pi^{(k)}$ , recursively defined via  $\pi^{(0)} = \epsilon$ , and, if  $\mathcal{L}_i = \emptyset$ , then  $\pi^{(i)} = \pi^{(i-1)} \cdot \langle o_i \rangle$ , else  $\pi^{(i)} = \pi^{(i-1)} \cdot \gamma \cdot \langle \overline{o_i} \rangle$ , where  $\gamma$  is some (arbitrary) sequencing of operators

$$\{get(L) \mid L \in \mathcal{L}_i \wedge \langle v_L/0 \rangle \in s_0[\pi_2][\pi^{(i-1)}]\}.$$

Finally, we set  $\pi'_2 := \pi_2 \cdot \pi_{2e}$ .

By Eqs. 17 and 18 in the definition of the budget reducing compilation, it is easy to verify that the above construction of  $\pi_{2e}$  ensures  $c_{\mathcal{L}^*}(\pi_{2e}) = c(\rho_{1e}) - \sum_{\langle v_L/1 \rangle \in s_2} lcost(L)$  and  $Q^{b_{\mathcal{L}^*}}(\pi_{2e}) = Q^b(\rho'_2)$ . In turn, by the properties of  $\pi_2$ , that implies  $Q^{b_{\mathcal{L}^*}}(\pi'_2) = Q^{b_{\mathcal{L}^*}}(\pi'_1)$  and  $c_{\mathcal{L}^*}(\pi'_2) = c_{\mathcal{L}^*}(\pi_2) + c_{\mathcal{L}^*}(\pi_{2e})$ .

Finally, since

$$c_{\mathcal{L}^*}(\pi_2) = c(\rho_2) - \sum_{\langle v_L/0 \rangle \in s_2} lcost(L)$$

and

$$c_{\mathcal{L}^*}(\pi_{2e}) = c(\rho_{1e}) - \sum_{\langle v_L/1 \rangle \in s_2} lcost(L),$$

we have

$$c_{\mathcal{L}^*}(\pi'_2) = c(\rho_2) + c(\rho_{1e}) - \sum_{L \in \mathcal{L}} lcost(L).$$

Thus, since  $c(\rho_1) \geq c(\rho_2)$  and  $\rho'_1 = \rho_1 \cdot \rho_{1e}$  is a valid plan for  $\Pi$ , we have

$$\begin{aligned} c_{\mathcal{L}^*}(\pi'_2) &\leq c(\rho_1) + c(\rho_{1e}) - \sum_{L \in \mathcal{L}} lcost(L) \\ &\leq c(\rho'_1) - \sum_{L \in \mathcal{L}} lcost(L) \\ &\leq b - \sum_{L \in \mathcal{L}} lcost(L), \end{aligned}$$

finalizing the proof that  $\pi'_2$  is a plan for  $\Pi_{\mathcal{L}^*}$  as in the claim.  $\square$



## Appendix B. Detailed Evaluation Results

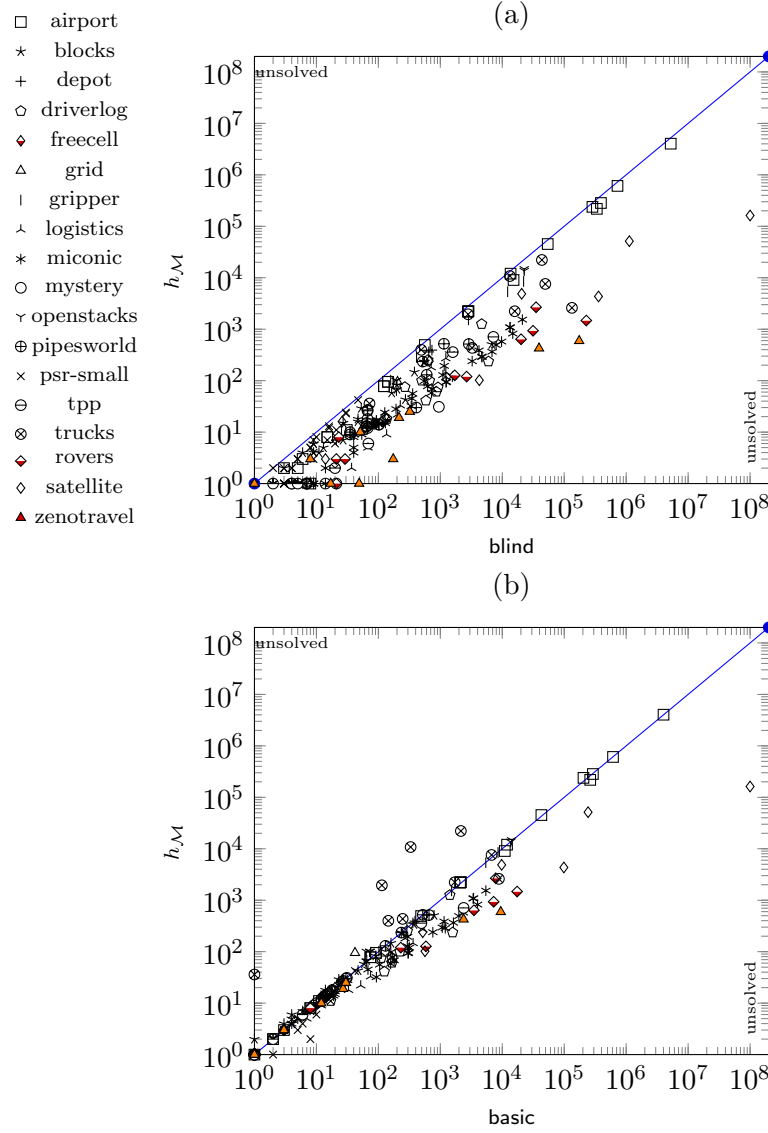


Figure 15: Restriction of the presentation in Figure 7, p. 19, to the tasks budgeted with 25% of the minimal cost required to achieve the entire set of sub-goals

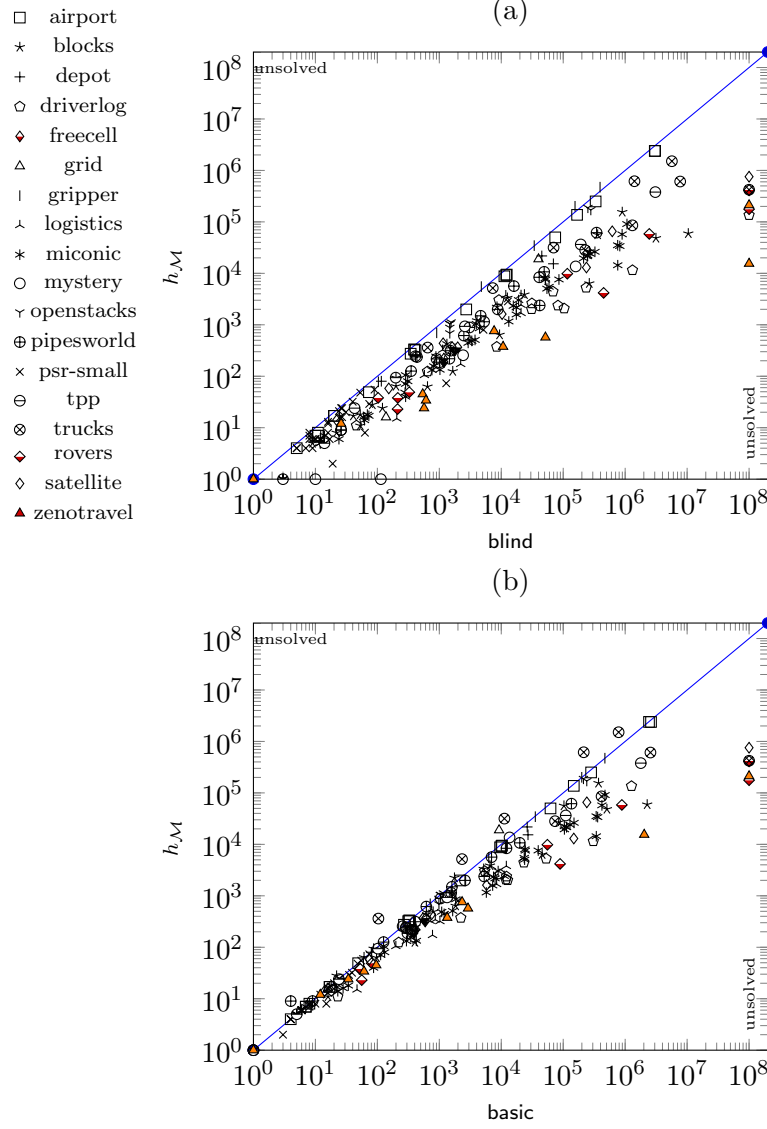


Figure 16: Restriction of the presentation in Figure 7, p. 19, to the tasks budgeted with 50% of the minimal cost required to achieve the entire set of sub-goals

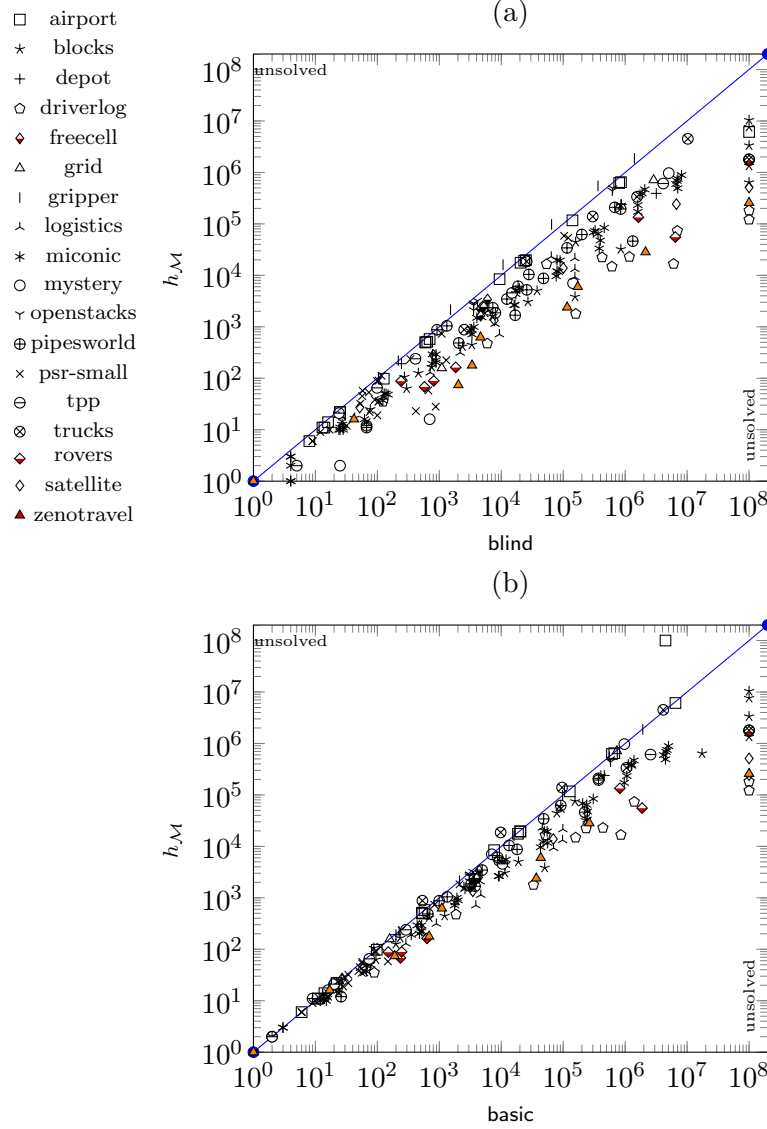


Figure 17: Restriction of the presentation in Figure 7, p. 19, to the tasks budgeted with 75% of the minimal cost required to achieve the entire set of sub-goals

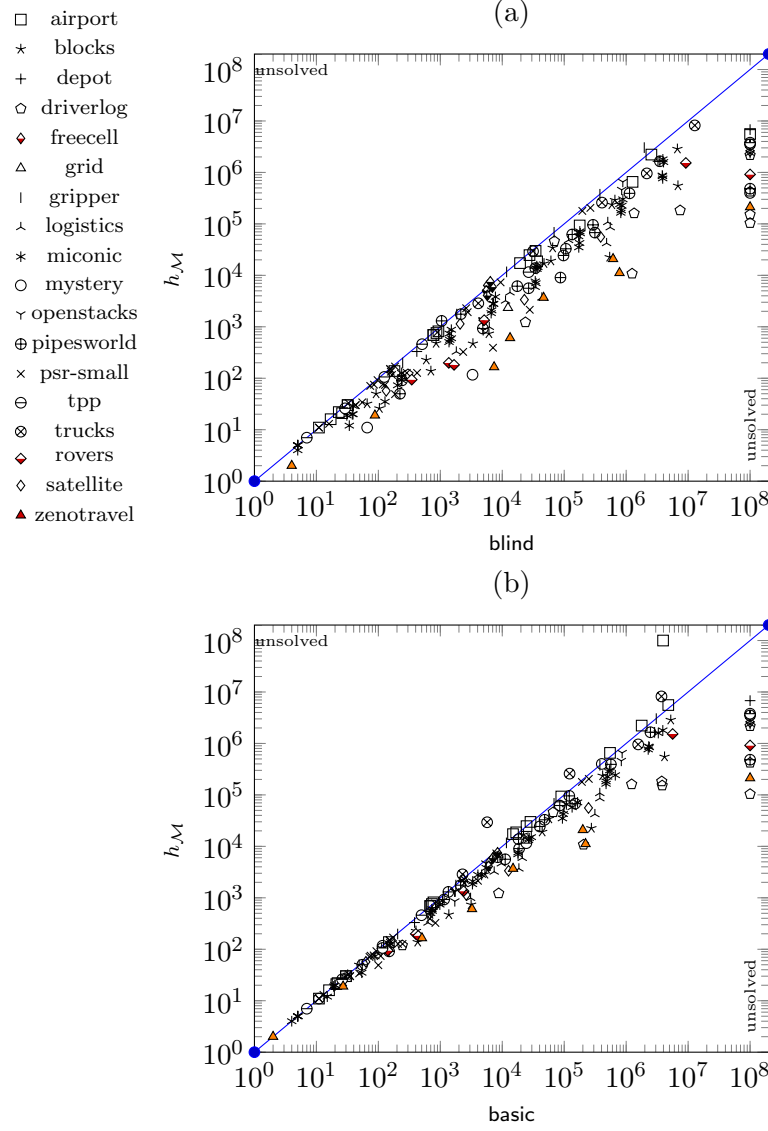


Figure 18: Restriction of the presentation in Figure 7, p. 19, to the tasks budgeted with 100% of the minimal cost required to achieve the entire set of sub-goals

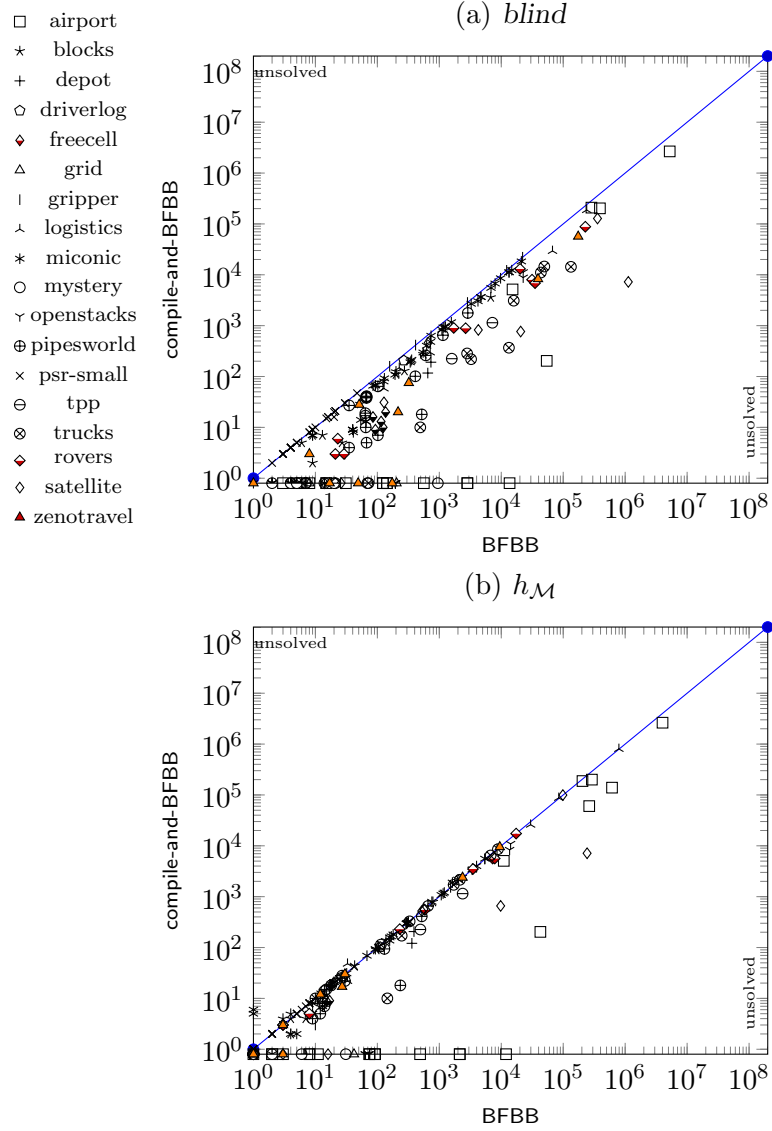


Figure 19: Restriction of the presentation in Figure 13, p. 41, to the tasks budgeted with 25% of the minimal cost required to achieve the entire set of sub-goals

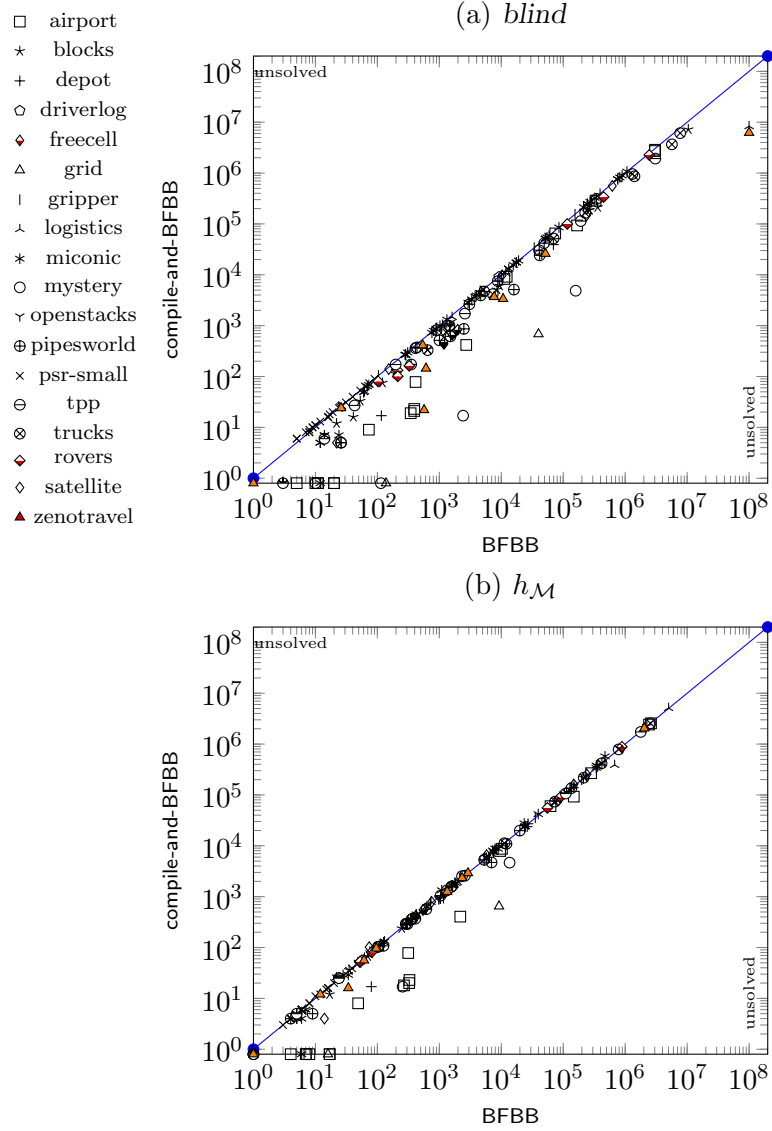


Figure 20: Restriction of the presentation in Figure 13, p. 41, to the tasks budgeted with 50% of the minimal cost required to achieve the entire set of sub-goals

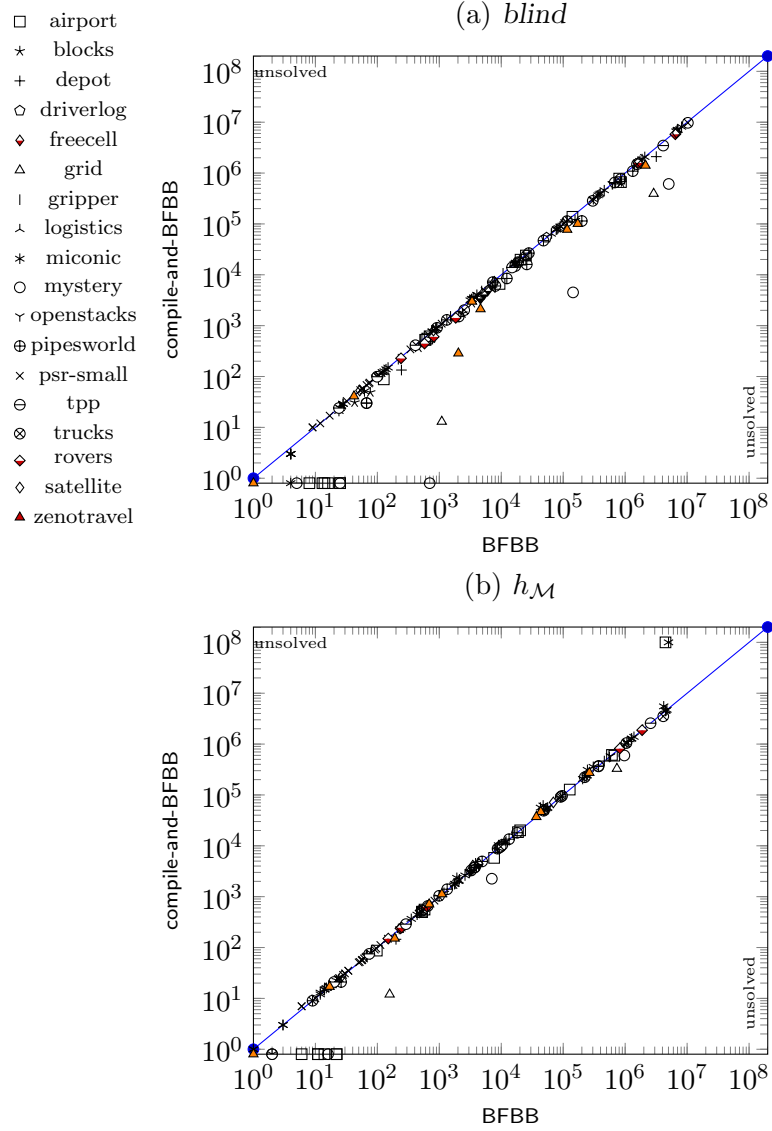


Figure 21: Restriction of the presentation in Figure 13, p. 41, to the tasks budgeted with 75% of the minimal cost required to achieve the entire set of sub-goals

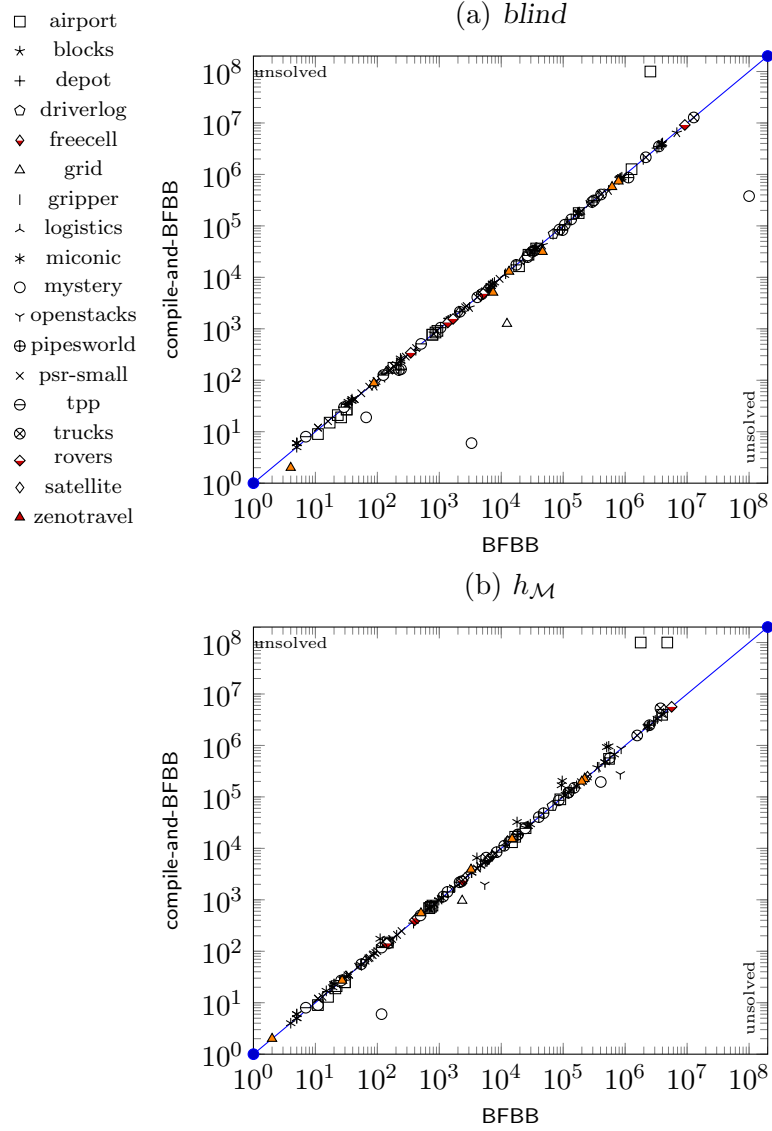


Figure 22: Restriction of the presentation in Figure 13, p. 41, to the tasks budgeted with 100% of the minimal cost required to achieve the entire set of sub-goals



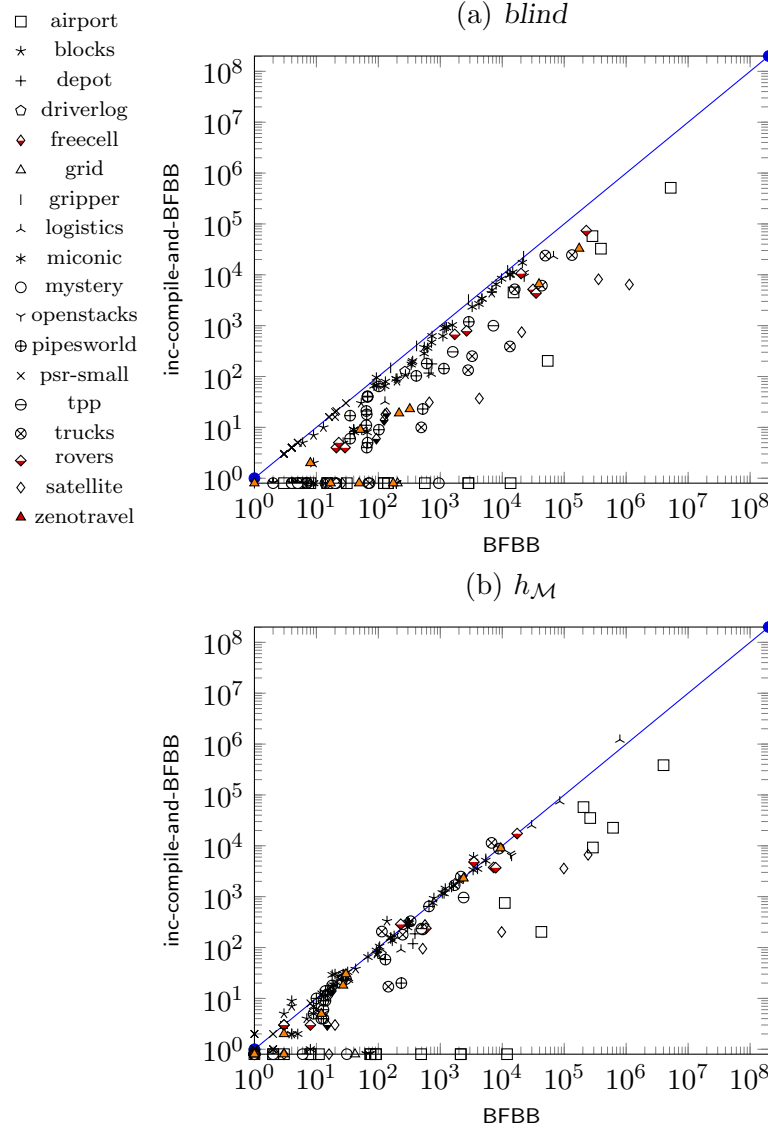


Figure 23: Restriction of the presentation in Figure 14, p. 42, to the tasks budgeted with 25% of the minimal cost required to achieve the entire set of sub-goals

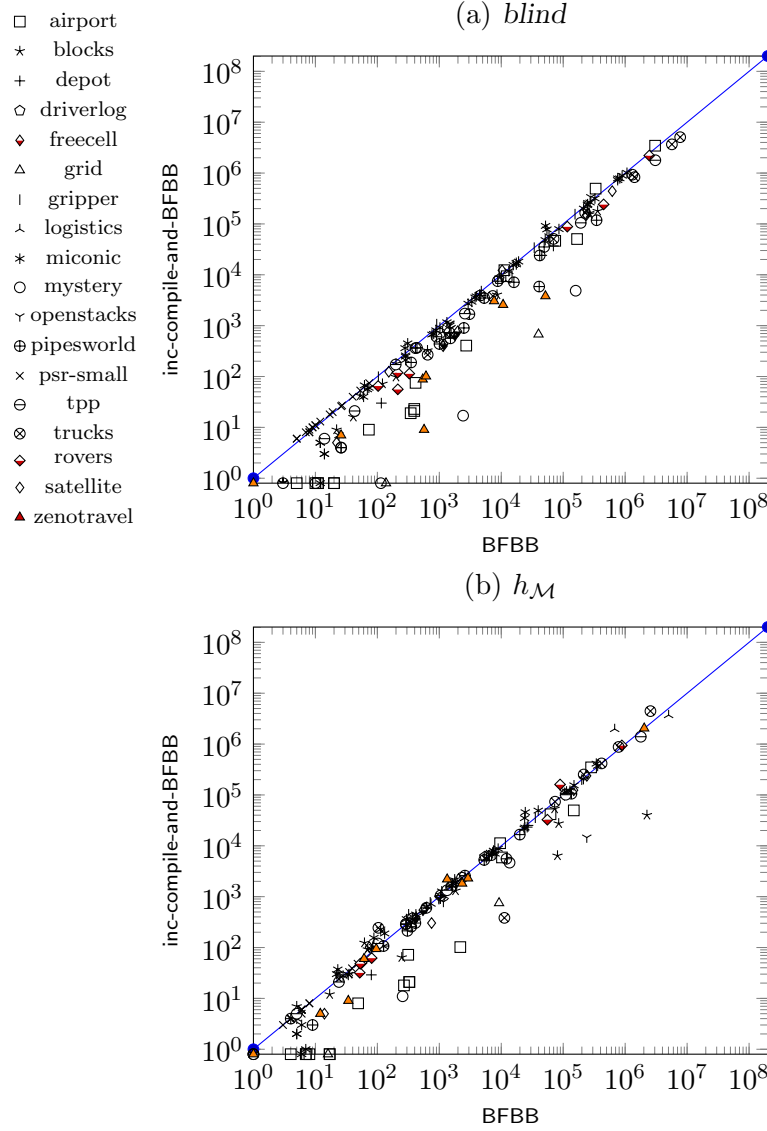


Figure 24: Restriction of the presentation in Figure 14, p. 42, to the tasks budgeted with 50% of the minimal cost required to achieve the entire set of sub-goals

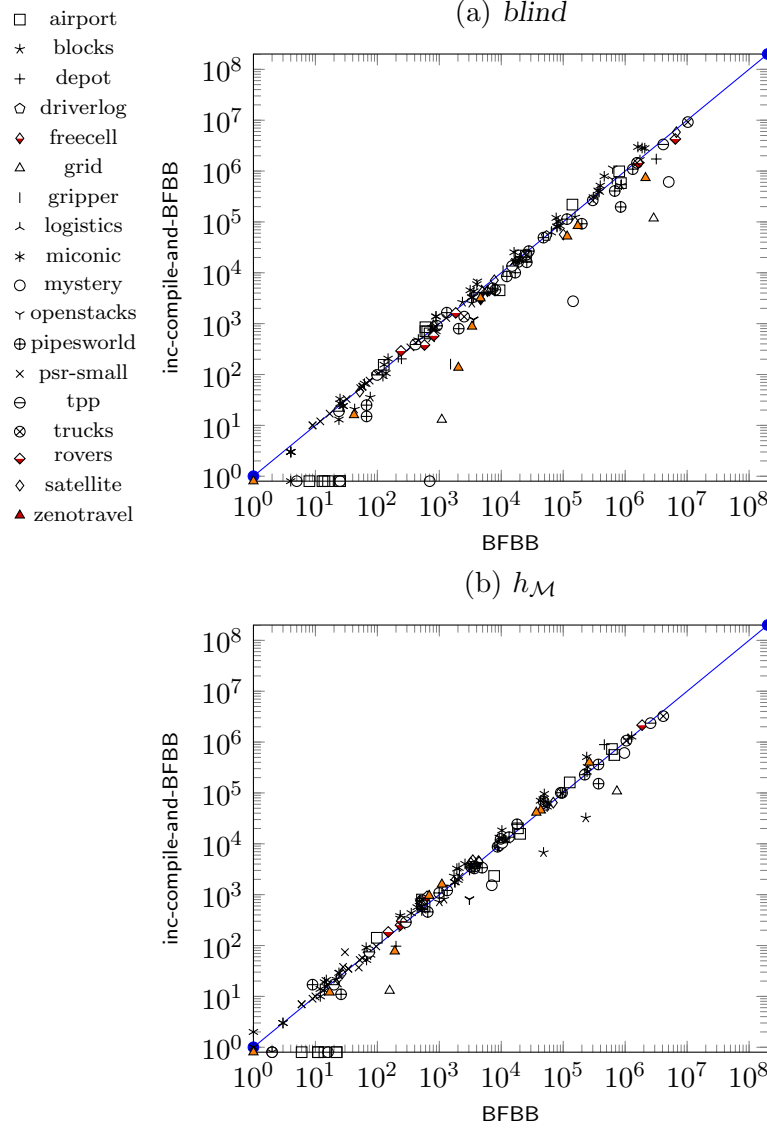


Figure 25: Restriction of the presentation in Figure 14, p. 42, to the tasks budgeted with 75% of the minimal cost required to achieve the entire set of sub-goals

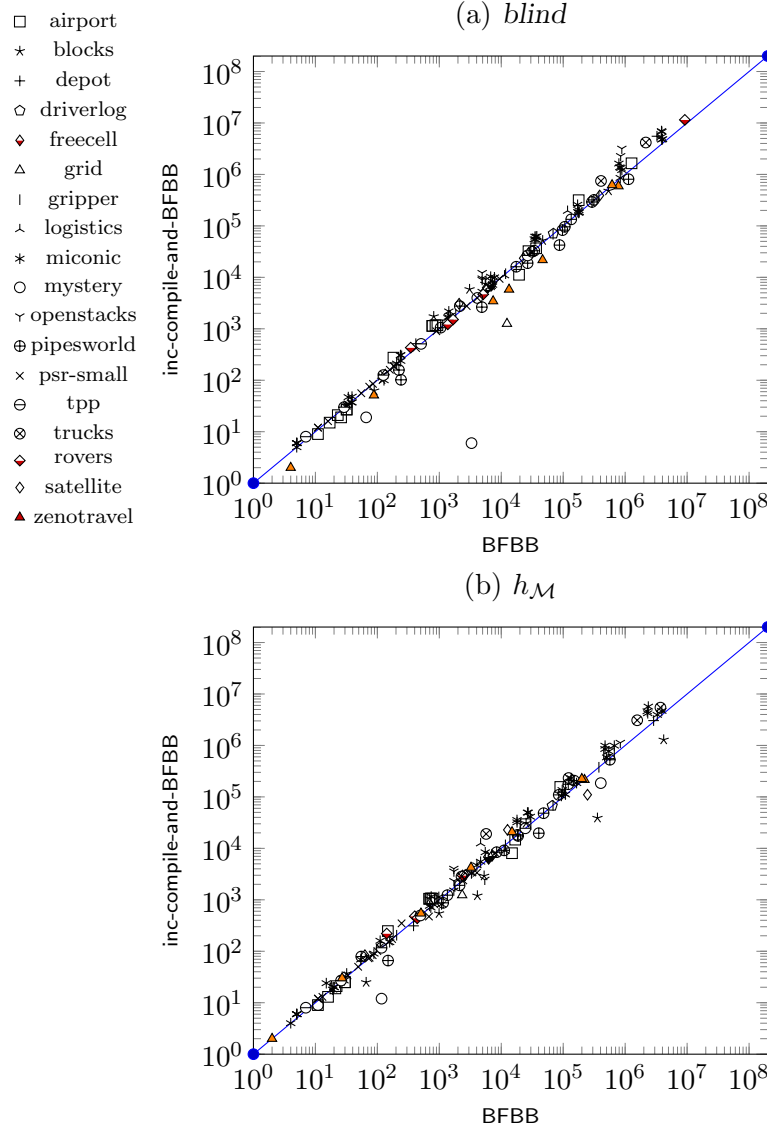


Figure 26: Restriction of the presentation in Figure 14, p. 42, to the tasks budgeted with 100% of the minimal cost required to achieve the entire set of sub-goals

## References

- Bäckström, C., & Klein, I. (1991). Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence*, 7(3), 181–197.
- Bäckström, C., & Nebel, B. (1995). Complexity results for SAS<sup>+</sup> planning. *Computational Intelligence*, 11(4), 625–655.
- Baier, J. A., Bacchus, F., & McIlraith, S. A. (2007). A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1808–1815.
- Baier, J. A., Bacchus, F., & McIlraith, S. (2009). A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6), 593–618.
- Benton, J., Coles, A. J., & Coles, A. I. (2012). Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 2–10.
- Benton, J., Do, M., & Kambhampati, S. (2009). Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, 173(5-6), 562–592.
- Benton, J., van den Briel, M., & Kambhampati, S. (2007). A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 34–41.
- Bonet, B. (2013). An admissible heuristic for SAS<sup>+</sup> planning obtained from the state equation. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2268–2274.
- Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2), 5–33.
- Bonet, B., & Geffner, H. (2008). Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *Artificial Intelligence*, 172(12-13), 1579–1604.
- Bonet, B., & Helmert, M. (2010a). Strengthening landmark heuristics via hitting sets. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pp. 329–334.
- Bonet, B., & Helmert, M. (2010b). Strengthening landmark heuristics via hitting sets. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 329–334, Lisbon, Portugal.
- Brafman, R. I., & Chernyavsky, Y. (2005). Planning with goal preferences and constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 182–191, Monterey, CA.
- Clarke, E., Grumberg, O., & Peled, D. (1999). *Model Checking*. MIT Press.
- Coles, A. I., Fox, M., Long, D., & Smith, A. J. (2008). Additive-disjunctive heuristics for optimal planning. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 44–51.

- Coles, A. J., Coles, A., Fox, M., & Long, D. (2013). A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research*, 46, 343–412.
- Coles, A. J., & Coles, A. I. (2011). LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 37–45.
- Cousot, P., & Cousot, R. (1992). Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4), 511–547.
- Dantzig, T. (1930). *Number: The Language of Science*. Macmillan.
- Desrochers, M., & Soumis, F. (1988). A generalized permanent labelling algorithm for the shortest path problem with time windows. *Information Systems and Operations Research*, 26, 191–212.
- Do, M. B., Benton, J., van den Briel, M., & Kambhampati, S. (2007). Planning with goal utility dependencies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1872–1878.
- Domshlak, C., Hoffmann, J., & Sabharwal, A. (2009). Friends or foes? On planning as satisfiability and abstract CNF encodings. *Journal of Artificial Intelligence Research*, 36, 415–469.
- Domshlak, C., Katz, M., & Lefler, S. (2012). Landmark-enhanced abstraction heuristics. *Artificial Intelligence*, 189, 48–68.
- Dudzinski, K., & Walukiewicz, S. (1987). Exact methods for the Knapsack problem and its generalizations. *European Journal of Operational Research*, 28, 3–21.
- Dvorak, F., & Barták, R. (2010). Integrating time and resources into planning. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 71–78.
- Edelkamp, S. (2001). Planning with pattern databases. In *Proceedings of the European Conference on Planning (ECP)*, pp. 13–24.
- Edelkamp, S. (2003). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, 20, 195–238.
- Fikes, R. E., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.
- Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning problems. *Journal of Artificial Intelligence Research*, 20, 61–124.
- Garey, M. R., & Johnson, D. S. (1978). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.
- Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.
- Gerevini, A., Haslum, P., Long, D., Saetti, A., & Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6), 619–668.

- Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20, 239–290.
- Gerevini, A., Saetti, A., & Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 172(8-9), 899–944.
- Grotschel, M., Lovasz, L., & Schrijver, A. (1981). The ellipsoid method and its consequences theorems in combinatorial optimization. *Combinatorica*, 1, 169–197.
- Handler, G., & Zang, I. (1980). A dual algorithm for the constrained shortest path problem. *Networks*, 10, 293–310.
- Haslum, P. (2013). Heuristics for bounded-cost search. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 312–316.
- Haslum, P., Bonet, B., & Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pp. 1163–1168, Pittsburgh, PA.
- Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, pp. 1007–1012.
- Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proceedings of the 15th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 140–149, Breckenridge, CO.
- Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning (ECP)*.
- Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*, Toulouse, France.
- Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26, 191–246.
- Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What’s the difference anyway?. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 162–169.
- Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 200–207.
- Hoffmann, J. (2003). The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research*, 20, 291–341.
- Hoffmann, J., Gomes, C. P., Selman, B., & Kautz, H. A. (2007). SAT encodings of state-space reachability problems in numeric domains. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1918–1923.
- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253–302.

- Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22, 215–278.
- Karp, R. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York.
- Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 1728–1733, Pasadena, CA.
- Katz, M., & Domshlak, C. (2010a). Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, 39, 51–126.
- Katz, M., & Domshlak, C. (2010b). Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174, 767–798.
- Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag Berlin.
- Keyder, E., & Geffner, H. (2009). Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36, 547–556.
- Koehler, J. (1998). Planning under resource constraints. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, pp. 489–493.
- Mirkis, V., & Domshlak, C. (2013). Abstractions for oversubscription planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 153–161, Rome, Italy.
- Mirkis, V., & Domshlak, C. (2014). Landmarks in oversubscription planning. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI)*, Prague.
- Nakhost, H., Hoffmann, J., & Müller, M. (2012). Resource-constrained planning: A Monte Carlo random walk approach. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 181–189.
- Pearl, J. (1984). *Heuristics - Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Pommerening, F., & Helmert, M. (2013). Incremental LM-Cut. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 162–170, Rome, Italy.
- Porteous, J., Sebastia, L., & Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 01)*, pp. 37–49.
- Punnen, A. P. (1992). K-sum linear programming. *The Journal of the Operational Research Society*, 43(4), 359–363.
- Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pp. 975–982, Chicago, IL.
- Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3 edition). Pearson.



- Sanchez, R., & Kambhampati, S. (2005). Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 192–201.
- Smith, D. (2004). Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 393–401.
- Thayer, J. T., & Ruml, W. (2011). Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 674–679.
- Thayer, J. T., Stern, R. T., Felner, A., & Ruml, W. (2012). Faster bounded-cost search using inadmissible estimates. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 270–278.
- van den Briel, M., Benton, J., Kambhampati, S., & Vossen, T. (2007). An LP-based heuristic for optimal planning. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 651–665.
- Yang, F., Culberson, J., Holte, R., Zahavi, U., & Felner, A. (2008). A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32, 631–662.

# Domain-independent Multi-agent Plan Repair

Antonín Komenda

*Department of Computer Science and Engineering  
Faculty of Electrical Engineering  
Czech Technical University in Prague  
Czech Republic*

Peter Novák

*Department of Software and Computer Technology  
Faculty of Electrical Engineering, Mathematics and Computer Science  
Delft University of Technology  
The Netherlands*

Michal Pěchouček

*Department of Computer Science and Engineering  
Faculty of Electrical Engineering  
Czech Technical University in Prague  
Czech Republic*

---

## Abstract

Achieving joint objectives in distributed domain-independent planning problems by teams of cooperative agents requires significant coordination and communication efforts. For systems facing a plan failure in a dynamic environment, arguably, attempts to repair the failed plan in general, and especially in the worst-case scenarios, do not straightforwardly bring any benefit in terms of time complexity. However, in multi-agent settings, the communication complexity might be of a much higher importance, possibly a high communication overhead might be even prohibitive in certain domains. We hypothesize that in decentralized systems, where frequent coordination is required to achieve joint objectives, attempts to repair failed multi-agent plans

---

*Email addresses:* `antonin.komenda@agents.fel.cvut.cz` (Antonín Komenda),  
`P.Novak@tudelft.nl` (Peter Novák), `micHAL.pechoucek@agents.fel.cvut.cz` (Michal Pěchouček)

should lead to lower communication overhead than replanning from scratch.

Here, we formally introduce the *multi-agent plan repair problem*. Building upon the formal treatment, we present the core hypothesis underlying our work and subsequently describe three algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem. Finally, we present an experimental validation, results of which confirm the core hypothesis of the paper. Our rigorous treatment of the problem and experimental results pave the way for both further analytical, as well algorithmic investigations of the problem.

*Keywords:* Multi-agent plan repair, Decentralized multi-agent planning, Communication complexity, Experimental evaluation

---

# Domain-independent Multi-agent Plan Repair

Antonín Komenda

*Department of Computer Science and Engineering  
Faculty of Electrical Engineering  
Czech Technical University in Prague  
Czech Republic*

Peter Novák

*Department of Software and Computer Technology  
Faculty of Electrical Engineering, Mathematics and Computer Science  
Delft University of Technology  
The Netherlands*

Michal Pěchouček

*Department of Computer Science and Engineering  
Faculty of Electrical Engineering  
Czech Technical University in Prague  
Czech Republic*

---

---

## 1. Motivation

Multi-agent planning based on classical planning is an approach to constructing control mechanisms for a team of possibly heterogeneous autonomous agents which compute and subsequently execute plans for the individual agents so as to achieve some joint team objective in the environment. When an agent is situated in a dynamic environment, occurrence of various unexpected events in the environment might lead to invalidation of the plan, a failure. A straightforward solution to this problem is to invoke a planning algorithm and compute a new plan from the state the failure occurred in to a state conforming with its original objective.

---

*Email addresses:* antonin.komenda@agents.fel.cvut.cz (Antonín Komenda),  
p.novak@tudelft.nl (Peter Novák), michal.pechoucek@agents.fel.cvut.cz (Michal Pěchouček)

In general, replanning in the case of a failure occurrence is a costly procedure, especially in terms of its time complexity. In many cases, however, a relatively minor fix to the original plan would resolve the failure possibly at a lower cost. Because it is not clear what exactly are the planning domains and types of dynamic environments which would benefit from such a repair approach, it can be argued that non-informed plan repair attempts can in many cases even raise the overall complexity of the approach in comparison to replanning. This would be due to futile attempts to repair the failed plan before inevitably falling back to replanning.

Plan repair can be seen as planning with re-use of fragments of the old plan. Even though there is a number of works, empirically demonstrating that plan repair in some domains performs better than replanning (e.g., [1, 2, 3]), Nebel and Koehler in [4] theoretically analyzed plan re-use (plan repair), and concluded that in general it does not bring any benefit over replanning in terms of computational time complexity. Taking into an account the performance of modern classical planners on modern hardware, the benefit of repairing failed plans would often be relatively low from the time-complexity perspective.

In situated multi-agent systems, however, the time complexity is often not of the primary importance. Consider application domains, such as e.g., undersea operations by teams of coordinated autonomous underwater vehicles. While the state-of-the-art technology allows to employ relatively powerful computers on board of such robots, the communication links are extremely constrained and expensive; wireless networks cannot be deployed and communication is performed mostly using acoustic signaling. In such applications, it is the *communication complexity* of the distributed planning algorithms which matters more than the time complexity. Consequently, employment of multi-agent plain repair techniques can provide a tangible benefit over replanning for a team of robots whose multi-agent plan fails.

Study of multi-agent planning system in previous literature [5, 6, 7] reveal a fact that local planning for individual agents has usually lower computational complexity than solving the global coordination problem. Therefore, at least intuitively, plan re-use techniques, which effectively simplify the coordination part of the problem, should improve the time complexity and, as the communication complexity is often tightly related to the time complexity, consequently lower the communication complexity as well.

The motivation for our research is the intuition that multi-agent plan repair, even though not always the fastest approach, should under specific con-

ditions generate lower communication overhead in comparison to replanning. The conditions correspond to the amount and frequency of minimal required coordination and the types of failures the environment generates. While the hypothesis is rather intuitive, investigation of the particular types of domains and the corresponding suitable repair algorithms deserves a deeper attention.

The contribution of the presented paper is threefold. Firstly, after introducing the general problem of multi-agent planning stemming from the formulation in [6], we give a rigorous treatment to the problem of multi-agent plan repair and formulate a notion of relative coordination frequency of a multi-agent planning problem. In turn, this formal approach allows us to state the core hypothesis of the presented research in a more formal and precise manner. Secondly, we propose three decentralized algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem including proofs of their correctness. Finally, we present experimental validation confirming the core hypothesis of the paper. The paper concludes with final remarks regarding the shortcomings of our approach and future outlooks in the here described line of research.

## 2. Multi-agent Planning

We treat the problem of multi-agent planning as an extension of the classical single-agent planning in the manner adapted from MA-STRIPS planning in [6]. We consider a number of *cooperative* and *coordinated* actors featuring distinct sets of capabilities (actions), which concurrently plan and subsequently execute their local plans so as to achieve a joint goal. An instance of a multi-agent planning problem is defined by i) an environment characterized by a state space, ii) a finite set of agents, each characterized by a set of primitive actions (or capabilities) it can execute in the environment, iii) an initial state the agents start their activities in and iv) a characterization of the desired goal states. The following formal restatement of the MA-STRIPS problem and our adaptations thereof constitute the preliminaries enabling us to state the core hypotheses of the paper in a formal manner, as well as provide the necessary background for the algorithms and their proofs introduced later in Section 3. The formal preliminaries build upon the original algorithm for the MA-STRIPS problem proposed by Brafman and Domshlak in [6].

A *state*  $s \subseteq \mathcal{L}$  is a set of atoms from a finite set of propositions  $\mathcal{L} = \{p_1, \dots, p_m\}$ . Given  $p \in s$ , we say that  $p$  *holds* in  $s$ , otherwise  $p$  *does not hold*

in  $s$ . In that sense, states are complete. That means, it cannot happen that there is a  $p \in \mathcal{L}$ , such that  $p$ 's validity in  $s$  is unknown.  $\mathcal{S} = 2^{\mathcal{L}} \cup \{\chi\}$  denotes the set of all states together with a distinguished state  $\chi \in \mathcal{S}$  denoting an undefined state.

A *primitive action* (*action*) an agent can perform in an environment is a tuple  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$ , where  $a$  is a unique action label and  $\text{pre}(a), \text{add}(a), \text{del}(a)$  respectively denote the sets of preconditions, add effects and delete effects of  $a$  taken from some  $\mathcal{L} = \{p_1, \dots, p_m\}$ .  $\text{Act}$  denotes the set of all actions and we furthermore assume there is a distinguished empty action  $\epsilon = \langle \emptyset, \emptyset, \emptyset \rangle \in \text{Act}$  with no preconditions and no effects. Whenever  $\text{pre}(a), \text{add}(a), \text{del}(a) \subseteq \mathcal{L}$ , we say that  $a$  is defined over  $\mathcal{L}$ .

We say that an action  $a$  is *applicable* in a state  $s$  iff  $\text{pre}(a) \subseteq s$ . An application of  $a$  is defined by the state transformation operator  $\oplus : \mathcal{S} \times \text{Act} \rightarrow \mathcal{S}$  so that  $s \oplus a = (s \cup \text{add}(a)) \setminus \text{del}(a)$  iff  $a$  is applicable in  $s$ . In the case  $a$  is not applicable in  $s$ ,  $s \oplus a$  results in a distinguished undefined state  $\chi$ . Note, we do not require that  $\text{add}(a) \cap \text{del}(a) = \emptyset$ , rather we simply assume that the effects negate each other strictly according to the definition of  $\oplus$ . Furthermore,  $\oplus$  is left-associative, hence we can write  $s \oplus a_1 \oplus \dots \oplus a_n$ .

An *agent*  $\alpha = \{a_1, \dots, a_n\}$  is characterized precisely by its capabilities, a finite repertoire of actions  $a_i \in \text{Act}$  it can perform in the environment.

**Definition 1** (MA-STRIPS). A *multi-agent planning problem* is a quadruple  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ , where

1.  $\mathcal{L}$  is a finite set of atoms;
2.  $\mathcal{A}$  is a set of *agents*  $\alpha_1, \dots, \alpha_n$  with actions defined over  $\mathcal{L}$ , featuring, besides the empty action  $\epsilon$ , otherwise mutually disjoint sets of actions. That is,  $\alpha_i \cap \alpha_j = \{\epsilon\}$ , whenever  $i \neq j$ ;
3.  $s_0 \in \mathcal{S}$  is an initial state; and finally
4.  $S_g \subseteq \mathcal{S}$  is a set of goal states.

From now on, given a set of agents  $\mathcal{A}$  as defined above,  $\text{Act} = \bigcup_{i=1}^n \alpha_i$  denotes the set of all actions which can be performed among the agents of the team  $\mathcal{A}$ , the team capabilities.

The definition of a MA-STRIPS multi-agent planning problem is adapted from the original formulation in [6]. Before formally defining the notion of a solution to a multi-agent planning problem, we first introduce a sequel of auxiliary notions.

Given an agent  $\alpha$ , a *single-agent plan*  $P$  is a sequence of actions  $a_1, \dots, a_k$ , s.t.,  $a_i \in \alpha$  for every  $i$ .  $P[i]$  denotes the  $i$ -th action in  $P$ , or  $P[i] = \epsilon$  in the case  $i$  is larger than the length of  $P$ , which in turn will be denoted  $|P|$ .

A team of agents  $\mathcal{A} = \alpha_1, \dots, \alpha_n$  can act in the environment concurrently. A joint action  $\mathbf{a} = \langle \text{pre}(\mathbf{a}), \text{add}(\mathbf{a}), \text{del}(\mathbf{a}) \rangle$  of the team is specified by  $\mathbf{a} = (a_1, \dots, a_n)$  a tuple of actions corresponding to the individual agents, that is,  $a_i \in \alpha_i$  for each  $i$ , its preconditions  $\text{pre}(\mathbf{a}) = \bigcup_{i=1}^n \text{pre}(a_i)$  and its effects  $\text{add}(\mathbf{a}) = \bigcup_{i=1}^n \text{add}(a_i)$  and  $\text{del}(\mathbf{a}) = \bigcup_{i=1}^n \text{del}(a_i)$ .  $\mathbf{a}[k]$  denotes the  $k$ -th action of  $\mathbf{a}$ . The notions of action applicability in a state  $s$ , as well as application of  $\mathbf{a}$  to  $s$  straightforwardly extend from the definitions for primitive actions, hence we can write  $s \oplus \mathbf{a}$ . Note, as we are building on top of the MA-STRIPS formalism, at this point we do not rule out, nor specifically handle joint actions in which the effects of individual agents' actions cancel out each other. In general, however, such considerations need to be tackled. Later on in this section, we comment on such joint actions in more detail.

**Definition 2** (multi-agent plan). Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a multi-agent planning problem with  $\mathcal{A} = \alpha_1, \dots, \alpha_n$ . A *synchronous multi-agent plan*  $\mathcal{P} = \{P_1, \dots, P_n\}$ , consisting of single agent plans  $P_1, \dots, P_n$  respectively constructed from actions of the agents  $\alpha_1, \dots, \alpha_n$  is a solution to  $\Pi$  if the plan  $\mathcal{P}$  satisfies the following:

1.  $\mathcal{P}$  is *well-formed*, i.e.,  $|P_i| = |P_j|$  for all  $i, j \leq n$ . Additionally,  $|\mathcal{P}| = |P_k|$  for every  $k \leq n$ , denotes the length of the multi-agent plan  $\mathcal{P}$ ;
2.  $\mathcal{P}$  is *feasible*, i.e., there exists a sequel of states  $s_1, \dots, s_m$ , s.t.  $m = |\mathcal{P}|$  and  $s_{i+1} = s_i \oplus \mathbf{a}_i$  with  $\mathbf{a}_i = (P_1[i], \dots, P_n[i])$  for all  $i < m$ ; and finally
3.  $\mathcal{P}$  reaches the goal  $S_g$ , i.e.,  $s_m \in S_g$ .

We also say that  $\mathcal{P}$  solves the problem  $\Pi$ . Finally,  $\text{Plans}(\Pi)$  denotes the set of plans which are solutions to a given multi-agent planning problem  $\Pi$ . Additionally,  $\mathcal{P}[k]$  denotes the joint action of the team in the step  $k$  and  $\mathcal{P}[k, i]$  denotes the primitive action of the agent  $i$  in the step  $k$ .

This notation allows us to introduce the following plan-matrix notation for a multi-agent plan  $\mathcal{P}$ , with  $a_{ij} = \mathcal{P}[i, j]$ , providing a more visual understanding of multi-agent plans



$$\mathcal{P} = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & & a_{m2} \\ \vdots & & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

We say that two multi-agent plans  $\mathcal{P}_1, \mathcal{P}_2$  are equal ( $\mathcal{P}_1 = \mathcal{P}_2$ ) iff they have the same length ( $|\mathcal{P}_1| = |\mathcal{P}_2|$ ) and for all  $i$  and  $j$  we have  $\mathcal{P}_1[i, j] = \mathcal{P}_2[i, j]$ .

A concatenation of two multi-agent plans  $\mathcal{P}_1$  and  $\mathcal{P}_2$  over the same agents  $\alpha_1, \dots, \alpha_n$  is defined as a plan  $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$ , where for each  $i$  and  $j$  we have  $\mathcal{P}[i, j] = \mathcal{P}_1[i, j]$  if  $i \leq |\mathcal{P}_1|$  and  $\mathcal{P}[i, j] = \mathcal{P}_2[i - |\mathcal{P}_1|, j]$  for  $i > |\mathcal{P}_1|$ . Note, concatenation of multi-agent plans is left-, as well as right- associative operation, so we can write  $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2 \cdot \dots \cdot \mathcal{P}_n$ .

Given a multi-agent plan  $\mathcal{P}$ ,  $\mathcal{P}[i..j]$  denotes a fragment of  $\mathcal{P}$  from the step  $i$  to the step  $j$ . More precisely,  $\mathcal{P}[i..j]$  is a fragment of  $\mathcal{P}$  iff there exist multi-agent plans  $\mathcal{P}_{prefix}$  and  $\mathcal{P}_{suffix}$ , such that  $\mathcal{P}_{prefix} \cdot \mathcal{P}[i..j] \cdot \mathcal{P}_{suffix} = \mathcal{P}$ . Finally,  $\mathcal{P}[i..\infty]$  denotes the  $i$ -th suffix of the plan  $\mathcal{P}$ , that is,  $\mathcal{P}[i..\infty] = \mathcal{P}[i..|\mathcal{P}|]$ .  $\mathcal{P}_1 \cdot \mathcal{P}_2$  is said to be a *decomposition* of a multi-agent plan  $\mathcal{P}$  iff  $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$ .

Given two multi-agent plans  $\mathcal{P}_1$  and  $\mathcal{P}_2$  we can define how different they are.  $diff(\mathcal{P}_1, \mathcal{P}_2)$  denotes the difference between  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , that is the overall number of primitive actions in  $\mathcal{P}_1$ , which do not correlate with the corresponding primitive actions in  $\mathcal{P}_2$  and *vice versa*.  $diff(\mathcal{P}_1, \mathcal{P}_2)$  corresponds to *Levensthein distance* [8], in literature also referred to as *edit-distance*, between two strings corresponding to the sequences of actions of the individual plans. Adaptation of the notion of Levensthein distance between two multi-agent plans corresponds to the number of atomic edits, that is insertion of an empty joint action, empty joint action deletion and individual action replacement, needed to transform one plan into the other. The cost of the atomic edits is assumed to be equal. This model of plan difference is also closely related to the MODDELINS modification problem for single-agent plans described in [4].

To introduce the MA-Plan algorithm for solving MA-STRIPS problems as formulated in [6], we finally need to distinguish between the public and private actions of individual agents. An action is *public* whenever its preconditions or effects involve atoms occurring in preconditions or effects of an action belonging to another agent of the team. The private actions are those, which are not affected by actions of the other agents.

Let  $\mathbf{atoms}(a) = \mathbf{pre}(a) \cup \mathbf{add}(a) \cup \mathbf{del}(a)$  and similarly  $\mathbf{atoms}(\alpha)$  be the

sets of atoms required or affected by the action  $a$  and the agent  $\alpha$  respectively. Given a multi-agent team  $\mathcal{A} = \alpha_1, \dots, \alpha_n$  with actions defined over the set of atoms  $\mathcal{L}$ , the set of *public* actions is defined as  $Act^{pub} = \{a \mid a \in \alpha_i \text{ and } \text{atoms}(a) \subseteq \mathcal{L} \setminus \text{atoms}(\alpha_i)\}$ . Consequently, the set of private actions is defined as  $Act^{priv} = Act \setminus Act^{pub}$ .

The distinction of actions to private and public turns out to be an important one. Since private actions do not depend, nor are dependencies of other actions performable by the team, planning of sequences of private actions can be implemented strictly locally by the agent the actions belongs to. In effect, the public actions become points of coordination among the multi-agent team members. The algorithm MA-Plan for solving a planning problem  $\Pi$  can be thought of in two interleaving stages until a suitable multi-agent plan is found: i) computation of a plan consisting exclusively of suitable coordination points of the agent team, and subsequently ii) computation of sequences of private actions filling the gaps between the public actions of each individual agent. While the second stage can be computed in a local manner by each individual agent without interactions with its peers, a truly decentralized multi-agent algorithm for the first stage requires a non-trivial amount of interaction between the agents.

One of the main contributions of the Brafman and Domshlak's paper [6] lies in the observation that the MA-Plan algorithm can be implemented by reduction of the first stage to a constraint satisfaction problem (CSP) (cf. e.g., [9]). In the CSP, each agent is represented by a single variable ranging over possible plans of the individual agent and two types of constraints:

**coordination constraint:** a sequence of joint actions  $\mathcal{P}$  (candidate multi-agent plan) corresponding to a multi-agent planning problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  satisfies the *coordination constraint* iff for every action  $a = \mathcal{P}[k, i]$  performed by the agent  $\alpha_i$  in the step  $k$  we have, that if  $a$  is a public action, then

- for every  $p \in \text{pre}(a)$ , there must exist  $a_p = \mathcal{P}[k_p, i_p]$ , such that  $p \in \text{add}(a_p)$  and  $0 < k_p < k$  (there is some previous action which causes  $p$  to hold), or  $p \in s_0$  in which case we set  $k_p = 1$ ; and
- for no  $k'$ , s.t.,  $k_p \leq k' \leq k$  there exists  $a' = \mathcal{P}[k', i']$ , such that  $p \in \text{del}(a')$  ( $p$  won't be invalidated between causing it in the step  $k_p$  and execution of  $a$  in the step  $k$ ).

---

**Algorithm 1** MA-Plan( $\Pi$ ):

---

**Input:** A multi-agent planning problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ .

**Output:** A multi-agent plan  $\mathcal{P}$  solving  $\Pi$ , if such exists.

```

 $\delta = 1$ 
loop
  construct  $\text{CSP}_{\Pi, \mathcal{A}}$ 
  if solve-csp( $\text{CSP}_{\Pi, \delta}$ ) then
    reconstruct a plan  $\mathcal{P}$  from a solution for  $\text{CSP}_{\Pi, \delta}$ 
    return  $\mathcal{P}$ 
  else
     $\delta = \delta + 1$ 
  end if
end loop

```

---

The constraint ensures that the dependencies of all the public actions occurring in the overall multi-agent plan are satisfied, possibly by actions performed in advance by other team members.

**internal planning constraint:** a sequence of joint actions  $\mathcal{P}$  corresponding to a multi-agent planning problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  satisfies the *internal planning constraint* iff for every agent, the corresponding single-agent planning problem with landmarks  $\{a \mid a = \mathcal{P}[k, i] \in \text{Act}^{\text{pub}}\}$  is solvable, meaning a single-agent planning algorithm is able to fill in the gaps between the public actions in the candidate multi-agent plan. The constraints ensure that each individual plan is locally executable by the particular agent.

Note, the formulation of the coordination constraint renders joint actions with  $\text{add}(\mathbf{a}) \cap \text{del}(\mathbf{a}) \neq \emptyset$  invalid. It is the non-strict inequalities in the condition 2 of the coordination constraint, together with the definition of public actions, which ensure the local consistency of joint actions.

Algorithm 1 lists the original multi-agent planning algorithm MA-Plan by Brafman and Domshlak in [6]. The algorithm iterates through CSP formulations of the planning problem according to  $\delta$ , informally the number of coordination points between the agents in the multi-agent team. That means,  $\delta$  determines the number of joint actions in a candidate multi-agent plan containing public actions. Filling the gaps between the individual single-agent

public actions, if possible, then gives rise to the overall multi-agent plan. In the case such a plan completion does not exist, the process continues by testing longer candidate plans, possibly not terminating in the case where no solution to the given multi-agent planning problem exists.

The original multi-agent planning algorithm assumes a centralized planning architecture. It is a centralized planning algorithm computing multi-agent plans for a team of agents which are supposed to be subsequently executed in a decentralized fashion. Our motivation is however a decentralized planning/plan repair algorithm followed by a decentralized plan execution.

In [10], Nissim, Brafman and Domshlak adapted the original blueprint algorithm from [6] to a distributed setting. The adaptation rests on formulating the multi-agent planning problem as a distributed constraint satisfaction problem instance (DisCSP) and subsequently utilizing a state-of-the-art DisCSP solver for solving it, plus managing the overhead involved in the resulting distributed algorithm. From now on, whenever we speak about the implementation of the multi-agent planning algorithm, we refer to its decentralized version as described in [10].

As mentioned in previous sections, the multi-agent planner from Nissim, Brafman and Domshlak [10] is built on a DisCSP solver and a centralized single-agent heuristic search planner. The algorithm used as the DisCSP solver is a customized Asynchronous Backtracking (ABT) solver [11] with Asynchronous Forward Checking [12] heuristics. In the planner, the best-first-search algorithm is employed with helpful action and landmark heuristics. The planner is a part of the FASTFORWARD planning suite [13]. The planning process passes four separate phases: i) centralized preparation of the DisCSP instance, ii) initialization of the solving process for the DisCSP in a special agent representing the goal requirements, iii) decentralized solving of the DisCSP problem and iv) decentralized finalization of the DisCSP solving process.

In the final decentralized phase the agents already know their local plans, given a multi-agent solution exists, and execute them in a distributed manner.

### 3. Multi-agent Plan Repair

Consider a multi-agent planning problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  and a plan  $\mathcal{P}$  solving  $\Pi$ . Furthermore, consider an environment in which, apart from the actions performed by the agents of the team  $\mathcal{A}$ , no other exogenous events occur. We say that such an environment is *ideal*, or *non-dynamic*.

The execution of  $\mathcal{P}$  in such an environment is failure-free and is uniquely determined by the set of states  $s_0, \dots, s_m$ , such that  $s_{i+1} = s_i \oplus \mathcal{P}[i]$  (cf. also Definition 2).

In dynamic environments, however, it can occur that in the course of execution of  $\mathcal{P}$ , the environment interferes and the execution of some action  $\mathcal{P}[i]$  from the plan  $\mathcal{P}$  does not result in precisely the state  $s_{i+1}$  as defined above. We could say that at step  $i$  an unexpected event occurred in the environment. For simplicity, we consider only unexpected events happening exclusively in the course of execution of some action (as if it took a non-zero time), not such which could occur while the agent is deliberating the execution (as if the deliberation was instantaneous).

Note that not all unexpected events in dynamic environments necessarily lead to problems with execution of the plan  $\mathcal{P}$ . However, there are at least two cases of such events, which can be considered a *plan execution failure*.

A *weak failure* of execution of the plan  $\mathcal{P}$  at step  $i$  w.r.t. the multi-agent planning problem  $\Pi$  is such, when the state  $s_f$  resulting from an attempt to perform the action  $\mathbf{a} = \mathcal{P}[i]$  does not satisfy some of the positive effects of  $\mathbf{a}$ , that is,  $\text{add}(\mathbf{a}) \not\subseteq s_f$ .

A *strong failure* of execution of the plan  $\mathcal{P}$  at step  $i$  w.r.t. the planning problem  $\Pi$  occurs whenever the  $i$ -th action of  $\mathcal{P}$  cannot be executed due to its inapplicability. It means, the execution of the plan up to the step  $i$  resulted in states  $s_0, s_1 \dots, s_i$ , possibly with some weak failures occurring in the course of execution of the plan fragment and  $\mathcal{P}[i]$  is not applicable in  $s_i$ .

The weak and the strong plan execution failures are, however, just two examples of a plan failure. There certainly are application domains in which weak failures can be tolerated as far as the goal state is reached after execution of the multi-agent plan. In practice, it makes the most sense to monitor for strong failures in system's evolution. Most weak failures either lead to a strong failure later on in the plan execution, or were irrelevant. Of course, except for the case when a weak failure leads to a future failure to reach a goal state, which happens, when some atom supposed to be included in a goal state fails to be effected by an action in the plan. There also might be domains in which other types of plan execution failures can occur, e.g., any change of the state not caused by the involved agents can be considered a failure as well. Thus, monitoring for weak, strong, or even other types of plan execution failures can strongly depend on the target application. To account for the range of various types failures, from now on, we only require that a plan execution monitoring process determines some plan execution failure at

a step  $i$  which results in some failed state  $s_f$ .

**Definition 3** (multi-agent plan repair). Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a multi-agent planning problem. A *multi-agent plan repair problem* is a quadruple  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ , where  $\mathcal{P}$  is a multi-agent plan solving the planning problem  $\Pi$ ,  $k$  is the step of  $\mathcal{P}$  in which its execution failed and  $s_f \in \mathcal{S}$  is the corresponding failed state.

A *solution to the plan repair problem*  $\Sigma$  is a multi-agent plan  $\mathcal{P}'$ , such that  $\mathcal{P}'$  is a solution to the planning problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ . We say that  $\mathcal{P}'$  *repairs*  $\mathcal{P}$  in  $s_f$ . In the case  $\text{Plans}(\Pi') = \emptyset$ , we say that the plan is *irreparable* given the failure occurring at the state  $s_f$ .

Given two multi-agent plans  $\mathcal{P}_1$  and  $\mathcal{P}_2$  both repairing a multi-agent plan  $\mathcal{P}$  for a problem  $\Pi$  in a state  $s_f$ , we say that  $\mathcal{P}_1$  is *preserving*  $\mathcal{P}$  more than  $\mathcal{P}_2$  iff  $\text{diff}(\mathcal{P}_1, \mathcal{P}) \leq \text{diff}(\mathcal{P}_2, \mathcal{P})$  and denote the relation by  $\mathcal{P}_1 \preceq \mathcal{P}_2$ . The *minimal repair of the multi-agent plan*  $\mathcal{P}$  is such a plan  $\mathcal{P}_{\min} \in \text{Plans}(\Pi')$ , which is minimal w.r.t. the mutual differences between the plans solving  $\Pi'$ . That is,

$$\mathcal{P}_{\min} \in \arg \min_{\mathcal{P}' \in \text{Plans}(\Pi')} \text{diff}(\mathcal{P}, \mathcal{P}')$$

Note, there might be several distinct minimal repairs of a given multi-agent plan.

In general, the multi-agent plan repair problem can be reduced to solving a modified multi-agent planning problem and thus gives rise to a straightforward plan repair algorithm based on *replanning* in two steps: i) construct the multi-agent replanning problem  $\Pi'$  as prescribed in Definition 3, and subsequently ii) utilize the **MA-Plan** algorithm to solve the problem  $\Pi'$ .

While the notion of minimal repair of multi-agent plans is based on the number of changes the repaired plan contains w.r.t. the original plan, also other metrics selecting distinguished plan repairs could be considered. We will discuss examples of such later in the paper.

The original motivation underlying this paper was the hypothesis that attempts to repair failed multi-agent plans lead to lower communication overhead than replanning. Clearly, not all planning problems could benefit from such a mechanism. Since we focus on multi-agent planning problems, which in a sense enforce coordination among the members of a multi-agent team, we need to provide an indication of which planning problems tend to benefit from the plan repairing approach. The following notion of *coordination frequency* formalizes the idea.

**Definition 4** (coordination frequency). Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a multi-agent planning problem with a solution  $\mathcal{P}$ . We say that  $\mathcal{P}$  is  $\delta$ -coordinated iff it contains at least  $\delta$  coordination points, that are, joint actions including at least one public actions of some individual agents. In the case  $\delta = 0$ , that is  $\mathcal{P}$  does not contain any public action, we say that  $\mathcal{P}$  is *uncoordinated*.

*Relative coordination frequency*  $cf(\mathcal{P})$  of a  $\delta$ -coordinated plan  $\mathcal{P}$  denotes the frequency of coordination point occurrence per single step in the plan and is defined as

$$cf(\mathcal{P}) = \frac{\delta}{|\mathcal{P}|}$$

*Relative coordination frequency*  $cf(\Pi)$  of a multi-agent planning problem  $\Pi$  denotes the minimal coordination frequency required to solve  $\Pi$  and is defined as

$$cf(\Pi) = \min_{\mathcal{P} \in Plans(\Pi)} cf(\mathcal{P})$$

The notion of relative coordination frequency of plans relates to the fractional amount of coordination corresponding to a single step in a plan execution. It straightforwardly extends to planning problems viewed as sets of plans solving them. We simply look for solutions requiring minimal relative amount of coordination required to solve the problem. The notion of relative coordination frequency allows for comparison and ordering of multi-agent planning problems according to the amount of coordination they minimally require for solving them. Informally, we will call problems with relatively low  $cf(\Pi)$  *loosely coordinated* and those with  $cf(\Pi)$  closer to 1 *tightly coordinated*. Note that a problem with  $cf(\Pi) = 0.5$  is still tightly coordinated, as for each coordination step, there is only one uncoordinated step. Multi-agent planning problems with  $cf(\Pi) = 0$  will be called *uncoordinated*.

Note, it still might be the case that even though a multi-agent planning problem can be solved without any coordination  $cf(\Pi) = 0$ , there still can exist coordinated plans in  $Plans(\Pi)$ , which are more efficient, e.g., shorter than the uncoordinated ones. For instance, consider a domain where the objective is that an agent  $A$  reaches a destination  $d$ . The agent  $A$  could move from its starting position to  $d$  on its own, albeit slowly and resulting in a relatively long plan. Alternatively,  $A$  could be transported quickly to  $d$  by another agent  $B$ . The latter plan would be shorter in terms of overall number of steps, but would require coordination. In result, repair of such a plan would be costlier in terms of communication overhead it incurs than the uncoordinated one, our main concern here.

The core hypothesis of the paper can be now stated more formally.

**Hypothesis 1.** *Multi-agent plan repair approaches producing more preserving repairs than replanning tend to generate lower communication overhead for tightly coordinated multi-agent problems.*

A crisper, though perhaps a more challenging version of the hypothesis would express the communication overhead in terms of the average communication complexity.

**Hypothesis 2.** *When applied to tightly coordinated planning problems, multi-agent plan repair algorithms producing more preserving repairs than replanning should feature a lower average communication complexity than replanning.*

As shown in [6],  $\delta$  turns out to play an important role in time-complexity analysis of the MA-STRIPS problem (cf. also Algorithm 1). Above, we hypothesize that it is the relative frequency of coordination points along the plans, which turns out to play a role in the communication complexity of plan repair. Plan repair for problems which require some coordination quite often along the plans should lead to re-use of fragments including relatively large number of coordination points, which do not have to be planned for again and thus leads to reduction of required communication in the repair process.

In the remainder of this paper, we approach resolution of Hypothesis 1. Treatment of Hypothesis 2 is beyond the scope of this paper and is left for future work.

In the following subsections, we describe three plan repairing algorithms. Sketches of ideas behind these algorithms were initially proposed in [14], in the following we provide novel descriptions and formalizations of the algorithms in full details.

### *3.1. Back-on-track Repair*

Unexpected event occurring in an environment can cause a failure in execution of a plan performed by a multi-agent team in that environment. The result would be that the overall state of the system won't be the one expected by an undisturbed plan execution at the particular time step. A straightforward idea to fix the problem is to utilize a multi-agent planner to produce a plan from the failed state to the originally expected state and



---

**Algorithm 2** Back-on-Track-Repair( $\Sigma$ )

---

**Input:** A multi-agent plan repair problem  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ , with  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  and a sequence of states  $s_0, \dots, s_m$ , a failure-free execution of  $\mathcal{P}$  would generate.

**Output:** A multi-agent plan  $\mathcal{P}'$  solving  $\Sigma$ , if a solution exists.

```
construct  $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \dots, s_m\} \cup S_g)$ 
if MA-Plan( $\Pi_{back}$ ) returns a solution  $\mathcal{P}_{back}$  then
    retrieve the state  $s_j$  of  $\mathcal{P}$  to which  $\mathcal{P}_{back}$  returns
    return  $\mathcal{P}' = \mathcal{P}_{back} \cdot \mathcal{P}[j \dots \infty]$ 
else
    return  $\mathcal{P}' = \chi$ 
end if
```

---

subsequently follow the rest of the original multi-agent plan from the step in which the failure occurred. The following multi-agent plan repair approach, coined *back-on-track* (BoT) repair, is inspired by this idea, in fact a slight generalization of it.

**Definition 5** (back-on-track repair). Let  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$  be a multi-agent plan repair problem and  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$  being the corresponding modified multi-agent replanning problem.

We say that a plan  $\mathcal{P}' \in \text{Plans}(\Pi')$  is a *back-on-track repair* of  $\mathcal{P}$  iff there is a decomposition of  $\mathcal{P}'$ , such that  $\mathcal{P}' = \mathcal{P}_{back} \cdot \mathcal{P}[i.. \infty]$  for some  $i \leq |\mathcal{P}|$ .

$\mathcal{P}' = \mathcal{P}_{back} \cdot \mathcal{P}[i.. \infty]$  is said to be a *proper back-on-track repair* iff  $|\mathcal{P}[i.. \infty]| > 0$ , i.e.,  $\mathcal{P}'$  preserves some non-empty suffix of  $\mathcal{P}$ .

Informally, the back-on-track approach tries to preserve a suffix of the original plan, prefix it with a newly computed plan  $\mathcal{P}_{back}$  starting in  $s_f$  and leading to some state along the execution of  $\mathcal{P}$  in the ideal environment. Note, all plans from  $\text{Plans}(\Pi')$  are back-on-track repairs of the original plan. The length of the preserved suffix of the original plan provides a handle on ordering of the plans according to the quality of repair. The longer the preserved suffix, the more preserving the plan is. On the other hand, even when the plan repair problem  $\Sigma$  is indeed solvable, there might not be any valid proper back-on-track repair of the original planning problem.

Algorithm 2 realizes a multi-agent plan repair procedure according to the back-on-track plan repair principle. Since the MA-Plan algorithm searches

for the shortest plan from the initial state to a goal state, the **Back-on-Track-Repair** computes plans which return back to the original one in the shortest possible way. The length of the overall repaired plan, however, depends also on the selection of a particular goal state  $s_g \in \{s_0, \dots, s_m\} \cup S_g$  of the planning problem  $\Pi_{back}$ . If the planning algorithm selects  $s_g$  according to an ordering from  $s_m$  to  $s_0$  and later on the remaining states from  $S_g$  for the same lengths of possible  $\mathcal{P}_{back}$  plans, the overall repaired resulting plan would also be the shortest, under a condition the result is a proper back-on-track repair.

The algorithm rests on invocation of the underlying multi-agent planner, hence its correctness relies on the correctness of the underlying planner. The following lemma states the soundness of Algorithm 2.

**Lemma 6** (Back-on-Track-Repair soundness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a multi-agent planning problem with agents situated in a dynamic environment in which the environment can interfere with the plan execution and let  $\mathcal{P}$  be a solution to  $\Pi$ . Let also  $s_f$  be a state resulting from an interference of the environment, a plan failure, at a step  $k$  of execution of the plan  $\mathcal{P}$ .  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$  denotes the corresponding multi-agent plan repair problem.*

*Unless the execution of **Back-on-Track-Repair**( $\Sigma$ ) finishes with the undefined plan  $\chi$ , a failure-free execution of the resulting plan  $\mathcal{P}'$  leads to some goal state of the original multi-agent planning problem  $\Pi$ .*

*Proof.* Follows straightforwardly from the construction of  $\Pi_{back}$  and that  $\mathcal{P}$  is a solution to  $\Pi$ . Either  $\mathcal{P}_{back}$  leads to some state along the ideal execution trace of the original plan  $\mathcal{P}$  and then the remainder of  $\mathcal{P}$  leading to the final state  $s_m \in S_g$  is reused, or a failure-free execution of  $\mathcal{P}_{back}$  would lead directly to some final state  $s_{end} \in S_g$  without reusing a part of  $\mathcal{P}$ .  $\square$

Furthermore, upon a failure of a plan execution, if there exists a plan from the failed state to a final state of the original multi-agent planning problem, the back-on-track algorithm is able to find a solution to the corresponding multi-agent plan repair problem.

**Lemma 7** (Back-on-Track-Repair completeness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ ,  $\mathcal{P}$ ,  $s_f$ ,  $k$  and consequently  $\Sigma$  be as assumed in Lemma 6.*

*If there exists a solution to the modified multi-agent planning problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ , then the execution of **Back-on-Track-Repair**( $\Sigma$ ) algorithm finishes and finds  $\mathcal{P}' \neq \chi$ , a solution repair of  $\mathcal{P}$ .*

*Proof.* Again, follows straightforwardly from construction of  $\Pi_{back}$  in the algorithm. Observe that if there is a solution plan to the problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ , then there also must exist at least the same solution to the modified planning problem  $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \dots, s_m\} \cup S_g)$ . That is, in the worst case, the back-on-track approach resorts to re-planning from scratch.  $\square$

The lemmas 6 and 7 establish how the back-on-track plan repair approach inherits its correctness from the underlying multi-agent planner. Note however, the algorithm is only *partially complete*, because in cases when there is no solution to a given multi-agent planning problem, it is not ensured that the algorithm MA-Plan terminates. Provided a totally complete multi-agent planning algorithm, directly replacing MA-Plan, total completeness of the Back-on-Track-Repair algorithm could be straightforwardly established by the lemmas above.

### 3.2. Simple Lazy Repair

The back-on-track multi-agent plan repair approach seeks to compute a new prefix to some suffix of the original plan and repair the failure by their concatenation. An alternative approach, coined *lazy*, attempts to preserve the remainder of the original multi-agent plan and close the gap between the state resulting from the failed plan execution and a goal state of the original planning problem.

Let  $s_f$  be the state resulting from a failure in execution of a multi-agent plan  $\mathcal{P}$  in a step  $k$ . We say that a sequence of joint actions  $\mathcal{P}'$  is an *executable remainder* of  $\mathcal{P}$  from the step  $k$  and the state  $s_f$  iff there exists a sequence of states  $s_k, \dots, s_{|\mathcal{P}|}$ , such that  $s_k = s_f$ ,  $s_{i+1} = s_i \oplus \mathcal{P}'[i - k + 1]$  and for every step  $i$  and every agent  $j$ , we have that  $\mathcal{P}'[i - k + 1, j] = \mathcal{P}[i, j]$  in the case  $\mathcal{P}[i, j]$  is applicable in the state  $s_i$  and  $\mathcal{P}'[i - k + 1, j] = \epsilon$  otherwise. The following definition provides a formal definition of the lazy approach.

**Definition 8** (simple lazy repair). Let  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$  be a multi-agent plan repair problem and  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$  be the corresponding modified multi-agent replanning problem.

We say that a plan  $\mathcal{P}' \in Plans(\Pi')$  is a *lazy repair* of  $\mathcal{P}$  iff there is a decomposition of  $\mathcal{P}'$ , such that  $\mathcal{P}' = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{lazy}$ , where  $\mathcal{P}_{[k..\infty]}$  is the executable remainder of  $\mathcal{P}$  from the step  $k$ , execution of which, starting from  $s_f$ , results in the state  $s_{lazy}$ , and  $\mathcal{P}_{lazy}$  is a solution to the multi-agent planning problem  $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$ .

---

**Algorithm 3** Lazy-Repair( $\Sigma$ )

---

**Input:** A multi-agent plan repair problem  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ , with  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ .

**Output:** A multi-agent plan  $\mathcal{P}'$  solving the problem  $\Sigma$ , if a solution exists.

construct  $\mathcal{P}_{[k..\infty]}$ , the executable remainder of  $\mathcal{P}[k..\infty]$  from the state  $s_f$

simulate execution of  $\mathcal{P}_{[k..\infty]}$  from  $s_f$  on, resulting in a final state  $s_{lazy}$

construct  $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$

$\mathcal{P}_{lazy} = \text{MA-Plan}(\Pi_{lazy})$

**return**  $\mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{lazy}$ , **unless**  $\mathcal{P}_{lazy} = \chi$  in which case **return**  $\chi$

---

Algorithm 3 realizes multi-agent plan repair based on the lazy repair approach described above.

Similarly to the back-on-track algorithm, Algorithm 3 inherits its correctness from the underlying multi-agent planner invoked internally.

**Lemma 9** (Lazy-Repair soundness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ ,  $\mathcal{P}$ ,  $s_f$ ,  $k$  and  $\Sigma$  be as assumed in the Lemma 6.*

*Unless the execution of Lazy-Repair( $\Sigma$ ) finishes with the undefined plan  $\chi$ , a failure-free execution of the resulting plan  $\mathcal{P}'$  leads to some goal state of the original multi-agent planning problem  $\Pi$ .*

*Proof.* In whichever state  $s_{lazy}$  a failure-free execution of the executable remainder of  $\mathcal{P}$  ends up, if existing, the solution plan to the problem  $\Pi_{lazy}$  will take the system from there to some final state corresponding to the original multi-agent planning problem  $\Pi$ . The executable remainder of  $\mathcal{P}$  from the state in which the failure occurred will get reused in the resulting plan.  $\square$

Unlike the back-on-track algorithm, the lazy approach is in general incomplete, as it might happen that the execution of the executable remainder of the original plan diverges to a state from which no plan to a goal state exists. The notion of the algorithm completeness has to be weakened to domains in which the agent team is at least capable to revert its own actions.

**Definition 10** (connected multi-agent planning domain). Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a multi-agent planning problem. Let also  $Act = \alpha_1 \times \dots \times \alpha_n$ , with  $\alpha_1, \dots, \alpha_n \in \mathcal{A}$ , and  $\mathcal{S} = 2^{\mathcal{L}}$ . We say that the planning problem induces a *connected planning domain* iff for every state  $s \in \mathcal{S}$  and a joint action  $\mathbf{a} \in Act$ , there exists a solution to the multi-agent planning problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s \oplus \mathbf{a}, s)$ , i.e, a plan  $\mathcal{P} = \mathbf{a}_1, \dots, \mathbf{a}_k$ , such that  $s = s \oplus \mathbf{a} \oplus \mathbf{a}_1 \oplus \dots \oplus \mathbf{a}_k$ .

In essence, the definition of connected multi-agent planning domain states that it is in the scope of capabilities of the multi-agent team  $\mathcal{A}$  to “undo”, or “revert”, effects of any of its own actions. Note, a single-agent version of the definition (with an omnipotent agent  $\bar{\alpha} = \bigcup_{\alpha_i \in \mathcal{A}} \alpha_i$ ) would also suffice, since we require that  $\epsilon \in \alpha$  for every  $\alpha \in \mathcal{A}$  and in a consequence any joint action of the team can be transformed into a corresponding multi-agent plan of length  $n$  with only a single agent acting in any given step of the plan.

The following lemma states that the **Lazy-Repair** algorithm is complete in connected planning domains.

**Lemma 11** (Lazy-Repair completeness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  inducing a connected multi-agent planning domain and we assume  $\mathcal{P}$ ,  $s_f$ ,  $k$ , as well as  $\Sigma$  are as in the Lemma 6. Let also  $s_{\text{lazy}}$  correspond to the state to which a failure-free execution of an executable remainder  $\mathcal{P}_{[k..\infty]}$  of  $\mathcal{P}[k..\infty]$  would lead.*

*If there exists a solution plan  $\mathcal{P}'$  to the multi-agent planning problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ , then the execution of **Lazy-Repair**( $\Sigma$ ) algorithm finishes and finds a plan  $\mathcal{P}^* \neq \chi$ , a solution repair of  $\mathcal{P}$ .*

*Proof.* Let  $\mathcal{P}_{[k..\infty]} = \mathbf{a}_{k+1}, \dots, \mathbf{a}_m$  be the executable remainder of  $\mathcal{P}[k..\infty]$  and let  $s_{k+1}, \dots, s_m$  be the states resulting from a failure-free execution of  $\mathcal{P}_{[k..\infty]}$ , i.e.,  $s_{j+1} = s_j \oplus \mathbf{a}_j$  for  $k+1 \leq j < m$ . Since the agent team acts in a connected planning domain, any of its actions is reversible, that is, its effects can be undone. Therefore for execution of each action  $\mathbf{a}_j$  above, there must exist a sequence of plans  $\mathcal{P}_{\mathbf{a}_j}^{\leftarrow}$ , each being a solution to the planning problem  $\Pi_{\mathbf{a}_j}^{\leftarrow} = (\mathcal{L}, \mathcal{A}, s_{j+1}, s_j)$ . Since we assume that there exists a plan  $\mathcal{P}'$  solution to the problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ , the plan  $\mathcal{P}^* = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{\mathbf{a}_{m-1}}^{\leftarrow} \cdot \dots \cdot \mathcal{P}_{\mathbf{a}_{k+1}}^{\leftarrow} \cdot \mathcal{P}'$  is a solution for the plan repairing problem  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ . That is, the solution plan first executes  $\mathcal{P}_{[k..\infty]}$ , the executable remainder of the original plan  $\mathcal{P}$  from the point of failure (as defined by the algorithm), then “undoes/reverts” effects of all the performed actions in  $\mathcal{P}_{[k..\infty]}$  and thus returns to the state  $s_f$ , and finally executes the plan  $\mathcal{P}'$ , existence of which we assume.  $\square$

The corollary of the line of reasoning leading to the proof of completeness of the lazy repair approach is that despite non-existence of irreversible environment interferences in some domains, it is the agent team whose actions can break the system evolution beyond repair. For illustration, even though it is not in the ability of the physical environment to push a robot over a

cliff, it is indeed in its own powers to jump from it during execution of an executable remainder of some, otherwise harmless plan, which failed shortly before. In such domains, the lazy approach has to be employed with caution.

To conclude, similarly to the back-on-track approach, Lemma 11 states only partial completeness of the Lazy-Repair algorithm the underlying multi-agent planner does not ensure termination.

### 3.3. Repeated Lazy Repair

In a dynamic environment, plan failures occur repeatedly. Even after a repair of a failed plan, it is possible for the repaired plan to fail again. In this situation both the back-on-track, as well as the lazy multi-agent plan repair algorithms lead to prolonging the really executed plan. In the case of the back-on-track approach, this is inevitable, since upon the repair, the subsequent plan execution process immediately processes the newly added plan fragment. In the case of the lazy repair, however, upon occurrence of another failure during execution of an already repaired plan, it is not always necessary to prolong the overall multi-agent plan. In the case a second failure occurs while still executing the plan fragment from the original plan preserved by the first repair, the suffix appended by the first repair can be discarded and replaced by a new plan suffix repairing the second failure, should it be necessary.

The following definition formally introduces *repeated lazy* (RLazy) *plan repair*, an extension of the lazy multi-agent plan repair approach introduced in Definition 8. For clarity, from now on, we refer to the lazy multi-agent plan repair introduced in the previous subsection as *simple lazy repair*.

**Definition 12** (repeated lazy repair). Let  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$  be a multi-agent plan repairing problem. Let also  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be the corresponding multi-agent planning problem with a solution of the form  $\mathcal{P} = \mathcal{P}' \cdot \mathcal{P}_{fix}$ . In the case this is the first failure encountered during execution of  $\mathcal{P}$ , we have  $|\mathcal{P}_{fix}| = 0$  and thus  $\mathcal{P} = \mathcal{P}'$ . Otherwise,  $\mathcal{P}$  is a simple lazy repair solution of some (previously solved) plain repair problem  $\Sigma_p = (\Pi, \mathcal{P}_p, s_{f_p}, k_p)$  composed of an executable remainder of  $\mathcal{P}_p$  (represented as  $\mathcal{P}'$ ) and a repair suffix  $\mathcal{P}_{fix}$ .

We say that  $\mathcal{P}''$  is a *repeated lazy repair* of  $\mathcal{P}$  iff

1.  $\mathcal{P}''$  is a simple lazy repair solution to  $\Sigma' = (\Pi, \mathcal{P}', s_{f_p}, k)$  in the case  $k \leq |\mathcal{P}'[k_p.. \infty]|$  (the failure occurred still within the executable remainder of  $\mathcal{P}_p[k_p.. \infty]$ ); or otherwise

2.  $\mathcal{P}''$  is a simple lazy repair solution to  $\Sigma' = (\Pi, \mathcal{P}, s_{f_p}, k)$ .

The repeated lazy repair leads to a straightforward extension of the simple lazy plan repair algorithm listed in Algorithm 3. The intuitive benefit of the straightforward application of the repeated lazy repair approach is that it should lead to shorter executed plans than would result from usage of the simple lazy repair. Consider a plan execution failure at step  $k_1$  of a plan  $\mathcal{P}$ . Simple lazy repair approach would fix it by appending a suffix  $\mathcal{P}_1$  resulting in the plan  $\mathcal{P}_{[k_1..\infty]} \cdot \mathcal{P}_1$ . Simple lazy repair of a second failure at a step  $k_2$  occurring still somewhere in the fragment  $\mathcal{P}_{[k_1..\infty]}$  would result in a solution  $\mathcal{P}_{[k_2..\infty]} \cdot \mathcal{P}_1 \cdot \mathcal{P}_2$  with a suffix  $\mathcal{P}_2$ , the solution to the second plan repair problem. Unlike that, upon occurrence of the second failure the repeated lazy repair discards the previously computed suffix  $\mathcal{P}_1$  and replaces it with a new suffix  $\mathcal{P}'_2$ , resulting in a repair solution  $\mathcal{P}_{[k_2..\infty]} \cdot \mathcal{P}'_2$ . The idea is that in many domains  $\mathcal{P}'_2$  should be shorter than the length of the combined suffix  $\mathcal{P}_1 \cdot \mathcal{P}_2$ . This could be especially beneficial in domains in which subsequent failures can even revert, or otherwise fix the ones occurring previously.

---

**Algorithm 4** Repeated-Lazy-Repair( $\Sigma$ )

---

**Input:** A multi-agent plan repairing problem  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$  with  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  and its solution  $\mathcal{P}$ . In the case  $\mathcal{P}$  is a lazy repair solution of a (previously solved) plain repair problem  $\Sigma_p = (\Pi, \mathcal{P}_p, s_{f_p}, k_p)$ , it takes the form  $\mathcal{P} = \mathcal{P}' \cdot \mathcal{P}_{fix}$ . Otherwise, in the case this is the first failure encountered,  $|\mathcal{P}_{fix}| = 0$ .

**Output:** A multi-agent plan solving  $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ .

```

if  $k \leq |\mathcal{P}'_{[k_p..\infty]}|$  then
  return Lazy-Repair( $(\Pi, \mathcal{P}', s_f, k)$ )
else
  return Lazy-Repair( $(\Pi, \mathcal{P}, s_f, k)$ )
end if

```

---

As with the previous two plan repairing approaches, we conclude the discourse with proofs of repeated lazy repair approach correctness.

**Lemma 13** (Repeated-Lazy-Repair soundness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ ,  $\mathcal{P}$ ,  $s_f$ ,  $k$  and  $\Sigma$  be as assumed in the Lemma 6.*

*Unless the execution of Repeated-Lazy-Repair( $\Sigma$ ) finishes with the undefined plan  $\chi$ , a failure-free execution of the resulting plan  $\mathcal{P}'$  leads to some goal state of the original multi-agent planning problem  $\Pi$ .*

*Proof.* Follows immediately from the soundness of the simple lazy repair approach in Lemma 9.  $\square$

**Lemma 14** (Repeated-Lazy-Repair completeness). *Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  inducing a connected multi-agent planning domain and we assume  $\mathcal{P}$ ,  $s_f$ ,  $k$ , as well as  $\Sigma$  are as in the Lemma 6.*

*If there exists a solution plan to the multi-agent planning problem  $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ , then the execution of Repeated-Lazy-Repair( $\Sigma$ ) algorithm finishes and finds a plan  $\mathcal{P}' \neq \chi$ , a solution repair of  $\mathcal{P}$ .*

*Proof.* Again, follows straightforwardly from the proof of completeness of the simple lazy repair algorithm. Note, the proof of Lemma 11 is independent of how exactly does the final state to which the executable remainder of the original plan leads to looks like, it can be arbitrary. Therefore, when we arbitrarily modify the executable remainder of the original plan, as in Algorithm 4, the proof still holds. That is, if there exists a plan  $\mathcal{P}''$  from  $s_f$  to some state in  $S_g$ , then in connected domains, there must exist at least the plan firstly executing the executable remainder of the original plan, subsequently a plan reverting its effects back to  $s_f$  and than finally performing the steps of  $\mathcal{P}''$ .  $\square$

#### 4. Multi-agent Plan Repairing Process

The multi-agent planning, executing, monitoring and repairing process has two phases. In the first phase, for a given domain a multi-agent plan is constructed using the MA-Plan algorithm. In the second phase, the plan is executed by the agents acting in a shared environment. In the course of the execution, the dynamics of the environment can interfere, possibly resulting in a failure of the executed plan. Since the plan execution is monitored by the multi-agent team, or a centralized observer, upon a failure detection a plan repair algorithm is invoked. In turn, to find particular repairing plans, the MA-Plan algorithm is invoked as specified by the plan repairing algorithms introduced in the previous section.

Scheme listed in Algorithm 5 shows the pseudo-code of the process. Since we assume complete information there is no difference between a decentralized and a centralized monitoring, hence for clarity, the algorithm instantiates the centralized version. As a consequence of the information completeness



---

**Algorithm 5** Plan execution and monitoring scheme.

---

**Input:** An initial multi-agent planning problem  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ .

---

```

 $\mathcal{P} = \text{MA-Plan}(\Pi)$ 
if  $\mathcal{P} = \chi$  then return fail
 $k = 1$ 

repeat
  agents perform  $\mathcal{P}[k]$ 
  if failure detected then
    retrieve the current state  $s$  from the environment
     $\mathcal{P} = \text{Repair}((\Pi, \mathcal{P}, s, k))$ 
     $k = 1$ 
  else
     $k = k + 1$ 
  end if
until  $\mathcal{P} = \chi$  or  $k > |\mathcal{P}|$ 

```

---

assumption, also the execution of the centralized initialization of the MA-Plan algorithm does not negatively affect the amount of communication in the system.

Before execution of each plan step, the algorithm checks whether a failure occurred and if so, invokes a plan repair algorithm. We do not explicitly articulate what a failure amounts to, since this can be application specific. Plausible options include checking for weak, or strong failures, i.e, validity of effects of the previously executed action, or validity of preconditions of the action to be executed next. Alternatively, in some applications it might be useful to check for any exogenous change of the current state not caused by the involved agents.

Finally, the algorithm accounts for the possibility that the plan repairing process can result in finding no solution to the failure. If that is the case, the algorithm finishes with the final plan equal to the undefined plan  $\chi$ . Note however, that Algorithm 5 does not necessarily terminate. Termination of the scheme relies on two factors. Firstly, it is the termination property of the underlying multi-agent planner invoked by the plan repair algorithms discussed in Section 3. Secondly, unless no repair to the occurred failure can be found, the algorithm terminates when it is capable to fully execute the com-

puted plan. In environments where failures can occur relatively frequently, it can however happen that the plan execution, monitoring and repair process would continually repair recurring failures sooner than the previous repair was fully executed. In a consequence, this would lead to a gradual prolongation of the executed plan so that it will never reach the end of its execution. Informally, for such domains, we could state that the Algorithm 5 terminates when the plan repair process generates sharply shorter repaired plans than is the time horizon in which the failures in the environment tend to occur. Results in [15] discuss steps towards a formal analysis of such a planning horizon and classification of various planning domains with respect to the frequency of failures occurring in an environment and the likelihood that an agent completes its plans without an interruption.

Instantiation of the execution, monitoring and repair scheme with the repeated lazy repair algorithm allows for an alternative plan execution model. The planning process invocation in the repair algorithm could be delayed until the execution of the preserved fragment of the original plan finishes. Such an approach could preserve significantly longer fragments of the original plan than instantiation of the original scheme in Algorithm 5 with the Repeated-Lazy-Repair algorithm. That is, upon a failure, instead of trying to repair the failed plan right away, as both the back-on-track and simple lazy plan repair algorithms invoked from the listed plan execution scheme would do, the system can simply proceed with execution of the remainder of the original plan and only after it finishes, the lazy plan repair is triggered. The approach simply ignores the plan failures during execution and postpones the repair the very end of the process, hence the “*lazy*” label for the two algorithms. In some domains, such an approach could significantly decrease the number of multi-agent planner invocations and in a consequence save a large amount of communication overhead.

## 5. Experimental Validation

To verify the Hypothesis 1, we conducted a series of experiments with implementations of the multi-agent plan repair algorithms described in the previous section. Below, firstly, we describe the experimental setup used for the experiments, then we interpret the data collected and finally, we revisit Hypothesis 1.

### 5.1. Experimental Setup

The experiments were based on a presented two-stage plan repairing algorithm, where we distinguish two types of plan failures: *action failures* and *state perturbations*. Both failure types are parametrized by a uniformly distributed probability  $P$ , which determines whether a simulation step fails, or not (a failure is generated only if there exists a plan to a goal, which obviate problems with irreversible actions). Both failure types are weak failures. That is, they are not handled immediately, but can preclude the plan execution and later result in a strong failure. Upon detection, a strong failure is handled by one of the plan repairing algorithms.

An *action failure* is simulated by not-execution of some of the individual agent actions from the actual plan step. The individual action is chosen according to a uniform probability distribution over the positions within a joint action. The individual failed action is then removed from the joint action and the current state is updated by the modified joint action.

The other simulated failure type, *state perturbation*, is parametrized by a positive non-zero integer  $c$ , which determines the number of state terms, which are removed from the current state, as well as the number of terms which are added to it. The terms to be added or removed are selected also randomly from the domain language according to a uniform distribution.

We implemented the experimental setup as a centralized simulator of the environment integrating the multi-agent domain-independent planner MA-Plan. The individual agents are initialized by the planner initialization process, together with a given planning problem instance. Each agent runs in its own thread and they deliberate asynchronously. The agents send peer-to-peer messages between themselves via the centralized simulator as well. The messages are sent by the integrated MA-Plan planner exclusively in the DisCSP phase.

The experiments were performed on *FX-8150 8-core* processor at 3.6GHz with *Java Virtual Machine* limited to 2.5GB of RAM. The individual measurements were parametrized by the plan failure probability  $P$  and each problem instance was executed 10 times with various value samples. The resulting data are, in the figures, presented with the natural distribution. The candlestick charts depict the differences between the minimal and the maximal measurements, together with the standard deviation. The accompanying charts represent the percentage ratio between the measured variable for the particular repairing method and replanning from scratch (normalized at the

100% level). The values presented in the result table are average values from the measurements of the same parametrization.

Since we are using the planner MA-Plan as a black-box algorithm, the relative proportion to the replanning approach bear a higher significance than the particular absolute numbers. Moreover, the general scalability of the proposed plan repairing algorithms stand or fall by the used planner. Therefore improvements in scalability of the presented plan repairing techniques should correspond to improvements of the used domain-independent multi-agent planner. As we stated before, we used current state-of-the-art planner. Improving such planner towards better scalability is an interesting open research question at this point.

### 5.2. Test Problems, Algorithms and Metrics

The experiments were conducted on four planning domains. Three of the domains originate in the standard single-agent IPC planning benchmarks [16]. Similarly to the evaluation of the MA-Plan implementation in [10], we chose domains, which are straightforwardly modifiable to the multi-agent setting: LOGISTICS (2–6 agents), ROVERS (2–4 agents), and SATELLITES (2–6 agents). Additionally, we have extended the set of IPC-based domains by a well known coordination domain COOPERATIVE PATHFINDING (2–4 agents).

Instances of the LOGISTICS problems are about transporting packages between locations by a fleet of heterogeneous transport vehicles. A representative example of a LOGISTICS problem  $\Pi^{log3}$ —used as one of the experiments—contains three agents controlling two trucks T1 and T2 and one airplane A. There are two cities, each with one storage depot (d1 and d2) and one airport (a1 and a2). The trucks can move  $m(\text{from}, \text{to})$  only within their cities, between one depot and one airport. The airplane can fly  $f(\text{from}, \text{to})$  among all airports in the environment, but cannot land at the depots. All vehicles can load  $l(\text{package}, \text{location})$  and unload  $u(\text{package}, \text{location})$  a package at a location. Initially, there is one package  $p$  at one of the depots and the goal is to transport it to the other depot in the other city. The trucks start at the depots and the airplane starts at one of the airports. A multi-agent plan solving this particular instance is  $\mathcal{P}^{log3} =$

$$\begin{array}{l} \text{A :} \\ \text{T1 :} \\ \text{T2 :} \end{array} \left( \begin{array}{ccccccccc} \epsilon & \epsilon & \epsilon & \overline{l(p, a1)} & f(a1, a2) & \overline{u(p, a2)} & \epsilon & \epsilon & \epsilon \\ l(p, d1) & m(d1, a1) & \overline{u(p, a1)} & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ m(d2, a2) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \overline{l(p, a2)} & m(a2, d2) & u(p, d2) \end{array} \right),$$

the coordination frequency for such problem is  $cf(\Pi^{log3}) = \frac{4}{9} = 0.\bar{4}$ , because  $\delta = 4$  and length of the plan  $|\mathcal{P}^{log3}| = 9$  and the presented plan  $\mathcal{P}^{log3}$  is

minimal from the perspective of the coordination points. For the context of the experiments, the LOGISTICS domain is tightly coordinated in that it requires relatively frequent coordination among the involved agents: airplanes and trucks need to wait for each other to load or unload the transported packages. The parallel version of the LOGISTICS domain (PAR) for 5 and 6 agents involves two parallel logistics sub-problems.

Problems of the ROVERS domain describe space exploration missions carried out by autonomous rovers equipped for three types of tasks: soil analysis  $s$ , rock analysis  $r$  and imaging  $i$ . The resulting data from the tasks has to be communicated  $c(s, r, i)$  back to the Earth in one data package over a communication channel available only for one of the rovers at a time. The data can be communicated only if they are prepared  $p(s/r/i)$ . The rock and soil analysis can be executed provided that the rover is at suitable position and has empty analytical store. The store can be emptied, if required. The rovers can move among predefined waypoints with a known information about the samples. Images can be taken only from suitable positions and with a camera calibrated and in a correct mode. An example problem  $\Pi^{rov3}$  used as one of the experiments has three fully equipped rovers R1, R2 and R3. A solution  $\mathcal{P}^{rov3} =$

$$\begin{array}{l} \text{R1 : } \left( \begin{array}{ccccccc} \dots & p(r1) & \dots & p(i1) & \dots & p(s1) & \epsilon & \overline{c(s1, r1, i1)} & \epsilon \\ \dots & p(r2) & \dots & p(i2) & \dots & p(s2) & \epsilon & \epsilon & \overline{c(s2, r2, i2)} \\ \dots & p(r3) & \dots & p(i3) & \dots & p(s3) & \overline{c(s3, r3, i3)} & \epsilon & \epsilon \end{array} \right) \\ \underbrace{\hspace{10em}}_{10 \text{ private actions}} \end{array}$$

has coordination frequency  $cf(\Pi^{rov3}) = \frac{3}{13} \doteq 0.23$  following the same procedure as presented in previous paragraph with LOGISTICS. Therefore, for the context of the experiments, the ROVERS domain is loosely coordinated in that it requires coordination only at the end of plans.

The SATELLITES domains describe planning for a set of independent satellites providing various types of deep space imagery  $i$  from the orbit. Each imaging instrument on board of a satellite has to be firstly turned to point at one of predefined target directions. Secondly, each imaging instrument has to be powered, switched on and calibrated before it can take an image  $t(i)$  in one of predefined modes. A solution of one of the experimental instance

$\Pi^{sat3}$  using three satellites S1, S2 and S3 is  $\mathcal{P}^{sat3} =$

$$\begin{array}{l} \text{S1 : } \left( \begin{array}{cc} \cdots & i(i1) \end{array} \right) \\ \text{S2 : } \left( \begin{array}{cc} \cdots & i(i2) \end{array} \right) \\ \text{S3 : } \left( \begin{array}{cc} \cdots & i(i3) \end{array} \right) \end{array} \underbrace{\hspace{1.5cm}}_{\text{3 private actions}}.$$

In this case the coordination frequency  $cf(\Pi^{sat3}) = \frac{0}{3} = 0$ , as there is no public action in an optimal plan. Therefore the domain is uncoordinated in that it does not need any coordination between the satellites acquiring images individually.

Finally, in the COOPERATIVE PATHFINDING domain, a team of robots move on a  $3 \times 3$  grid (positions  $x1y1$  to  $x3y3$ ), where only a single robot can occupy one cell. The goal for the robots is to move  $m(\text{from}, \text{to})$  to initial positions occupied by the other robots in the initial state. A representative problem  $\Pi^{cp3}$  contains three robots R1, R2 and R3 and a solution plan  $\mathcal{P}^{cp3}$  is

$$\begin{array}{l} \text{R1 : } \left( \begin{array}{cc} \overline{m(x1y2, x1y1)} & \overline{m(x1y1, x2y1)} \\ \overline{m(x2y1, x3y1)} & \overline{m(x3y1, x3y2)} \end{array} \right), \\ \text{R2 : } \left( \begin{array}{cc} \overline{m(x2y1, x3y1)} & \overline{m(x3y1, x3y2)} \\ \overline{m(x3y2, x2y2)} & \overline{m(x2y2, x1y2)} \end{array} \right), \\ \text{R3 : } \left( \begin{array}{cc} \overline{m(x1y2, x1y1)} & \overline{m(x1y1, x2y1)} \\ \overline{m(x2y1, x3y1)} & \overline{m(x3y1, x3y2)} \end{array} \right), \end{array}$$

consequently the coordination frequency  $cf(\Pi^{cp3}) = \frac{2}{2} = 1$  as each action in an optimal plan is public and therefore the domain represent a fully coordinated problems.

To evaluate validity of Hypothesis 1, the multi-agent planning problems were tested on the experimental setup against a plan repair algorithm implementing replanning from scratch and two of the repair algorithms **Back-on-Track-Repair** (BoT repair, Algorithm 2) and **Repeated-Lazy-Repair** (RLazy repair, Algorithm 4) introduced in the previous section.

Efficiency problems of the original MA-Plan implementation (in [10]) limited the experiments to plans with maximally six landmarks, coordination points, per agent. Additionally, the **Back-on-Track-Repair** algorithm could not leverage disjunctive goal form (cf. construction of  $\Pi_{back}$  in Algorithm 2) and this was emulated by an iterative process testing all term conjunctions in a sequence and thus resulting in multiple runs of the DisCSP solver instead of a single run with disjunctive goal.

We used four metrics to evaluate the measurements:

**execution length** is the overall number of joint actions the experimental setup executed,

**planning time** was the measured cumulative time consumed by the underlying MA-Plan planner used for generating initial and repairing plans,

**repairing time** is the overall time spent in MA-Plan invocations minus the first planning process of the initial plan; and finally,

**communication** corresponds to the number of messages and communication volume in bytes passed between the agents during the planning or plan repair process. That is messages generated by the DisCSP solver in the MA-Plan planner.

### 5.3. Results and Discussion

The first batch of experiments directly targets validation of Hypothesis 1:

*Multi-agent plan repair is expected to generate lower communication overhead in tightly coordinated domains.*

LOGISTICS and COOPERATIVE PATHFINDING, as tightly and fully coordinated domains with dynamics of the simulated environment modeled as action failures, are suitable experiments to provide required insight. Table 1 shows results for a fixed failure probability  $P = 0.3$  and Figures 1, 2 and 3 depict the results of the experiment for 3 agents LOGISTICS with variable probability  $P$ .

The highlighted results for 4-agent LOGISTICS in the table shows that the communication overhead generated by the **Repeated-Lazy-Repair** (RLazy) algorithm is at 25% of that generated by the replanning approach. For 4-agent COOPERATIVE PATHFINDING, the communication overhead generated by the **Back-on-Track-Repair** (BoT) is at 18% of that generated by the replanning. Additionally, the communication overhead decreases with the increasing number of agents in the problems. That means, the plan repairing algorithms scale better than replanning from scratch. The trends in Figures 1, 2 and 3 for LOGISTICS domain show, that the results are also valid for higher values of  $P$ . Furthermore the overhead decreases with increasing failure probabilities. The communication overhead generated in the experiment for various probabilities  $P$  by the **Back-on-Track-Repair** algorithm is, over all the measured probabilities, on an average at 59% (36% at best) of that

Domain	Agents	Repairing time [ms]			No. of messages [-]			Communication [kB]			Exec. length [-]		
		BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan
LOGISTICS	2	115.3	116.1	145.6	13.9	8.2	10.9	2.0	1.7	2.3	8.8	14.3	10.7
	3	178.0	149.6	257.2	33.7	18.0	59.5	5.0	3.6	6.2	13.2	18.7	15.9
	4	266.2	162.1	479.3	89.5	29.1	114.7	13.3	<b>6.6</b>	<b>26.5</b>	15.5	21.5	18.1
LOGISTICS (PAR)	5	73.7	74.0	81.3	23.2	21.8	22.5	4.4	4.4	5.0	13.5	14.6	14.9
	6	126.0	84.1	110.7	49.6	32.5	60.9	9.3	6.8	9.6	11.4	12.4	12.0
COOP. PATHFINDING	2	23.9	115.6	93.2	2.4	2.2	2.2	0.6	0.6	0.6	2.8	3.2	2.6
	3	28.1	261.5	374.6	12.9	20.0	12.7	0.7	4.5	3.4	2.4	3.1	3.4
	4	29.4	19568.6	6529.8	20.0	14k	19.1	<b>0.9</b>	3002.0	<b>5.1</b>	2.3	4.0	3.3
ROVERS	2	381.3	179.8	249.8	13.0	7.4	10.0	2.7	1.9	2.7	14.7	20.3	14.5
	3	374.5	300.6	489.9	15	13.6	22.4	3.3	3.6	6.3	12.5	19.5	14.5
	4	798.3	634.4	650.5	42.5	30.0	29.1	7.6	8.3	8.9	15.3	22.1	13.6
SATELLITES	2	80.3	67.5	67.5	6.2	4.9	3.8	1.3	1.1	0.9	5.8	7.5	5.1
	3	126.9	81.9	139.9	15.8	7.6	15.3	2.9	1.8	3.7	6.9	7.0	6.5
	4	139.2	144.4	176.5	21.8	14.5	18.2	3.8	3.6	4.7	6.7	6.9	5.6
	5	154.5	232.9	222.3	30.0	17.7	25.9	5.6	4.8	7.3	5.7	6.7	5.5
	6	1452.8	48093.9	23027.7	57.3	32.5	38.2	11.6	9.4	11.7	6.9	7.1	5.7

Table 1: Results of experiments for all domains with probability  $P = 0.3$  and action failures. The highlighted cells are the best results for a particular domain and a particular metrics. The bolded results distinctively support the core hypothesis of the paper.



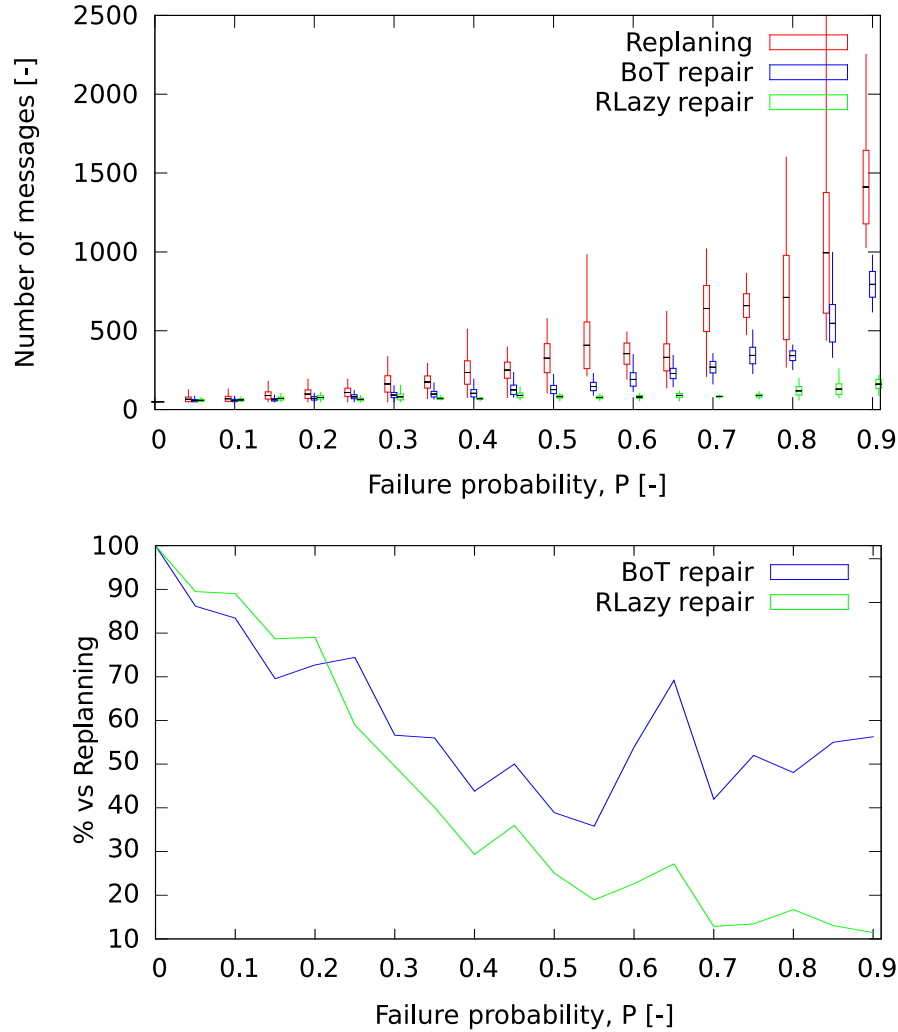


Figure 1: Experimental results of the communication metrics for LOGISTICS domain with 3 agents and action failures.

generated by the replanning approach. The Repeated-Lazy-Repair algorithm performed even better and on average produced only 43% (11% at best) of the communication overhead generated by the replanning algorithm. In a consequence, the experiments *strongly support* our hypothesis.

The overall time spent in the planning phase (used by the MA-Plan algorithm) by the plan repair algorithms echoes the results for the communication overhead. Plan repairing scales better with higher numbers of agents in both

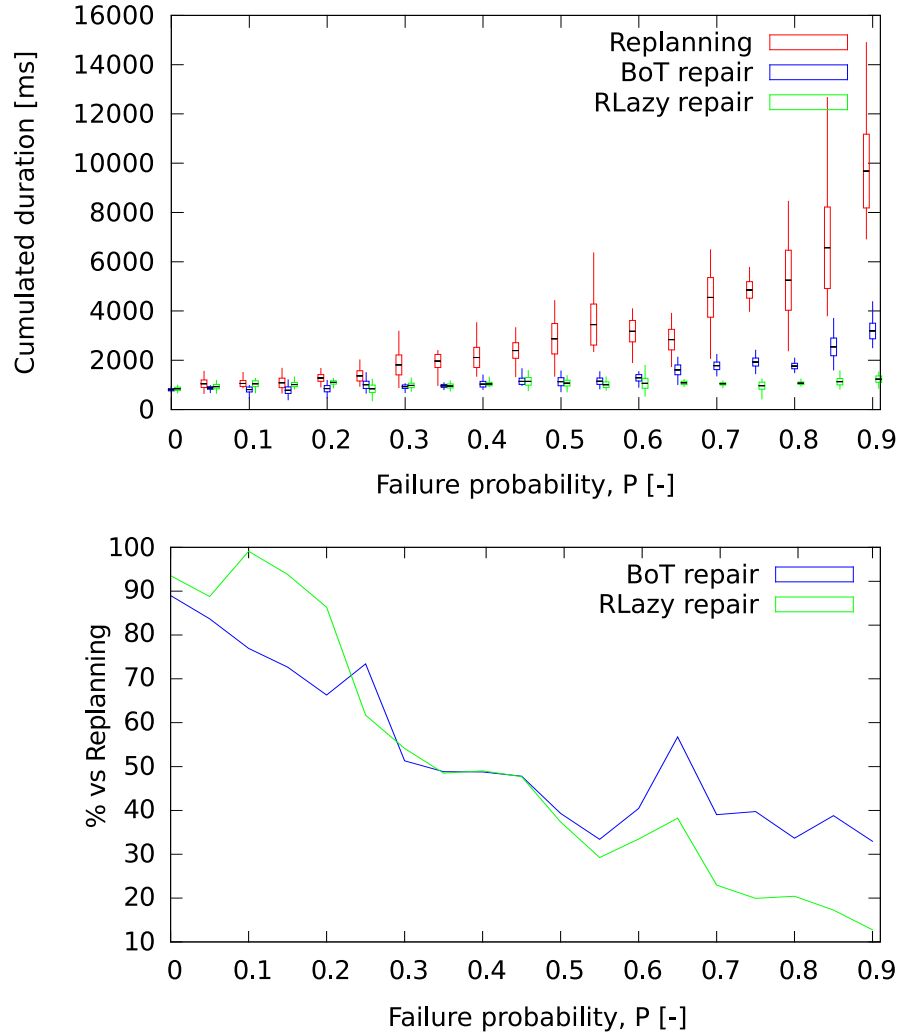


Figure 2: Experimental results of the planning time metrics for LOGISTICS domain with 3 agents and action failures.

LOGISTICS and COOPERATIVE PATHFINDING. On average, over all the measured probabilities  $P$  in 3-agent LOGISTICS, the computational efficiency was at 54% (34% at best) and at 51% (12% at best) for **Back-on-Track-Repair** and **Repeated-Lazy-Repair** respectively in comparison to replanning. Figure 1 depicts these results.

The second batch of experiments focused on boundaries of validity of the positive result presented above. In particular, we validated the condition on

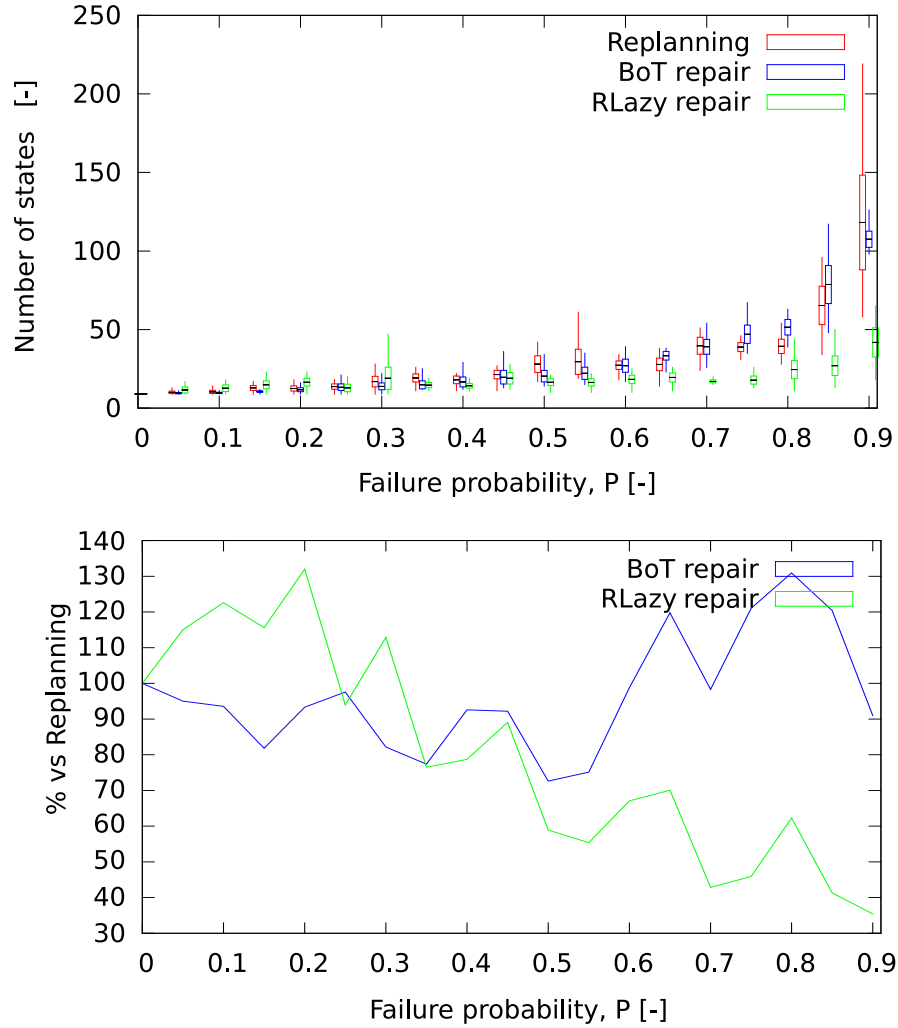


Figure 3: Experimental results of the execution length metrics for LOGISTICS domain with 3 agents and action failures.

the coordination tightness and feasibility of failures. The auxiliary hypothesis we validated states:

*With decreasing coordination frequency of the planning domain, the communication efficiency gains of repairing techniques should decrease. For loosely coordinated domains the communication efficiency of plan repair should be on-par with that of the replanning approach.*

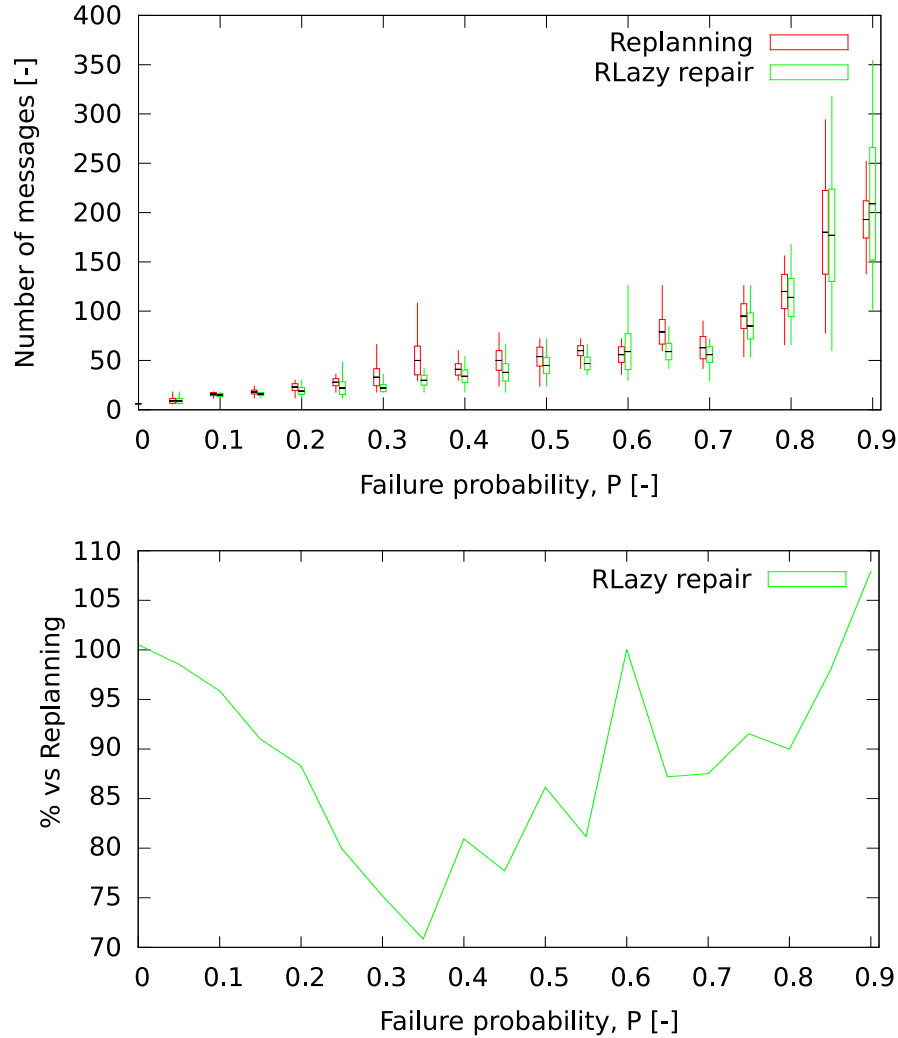


Figure 4: Experimental results for ROVERS domain with 3 agents and action failures.

To validate the auxiliary hypothesis we ran experiments with ROVERS as a loosely coordinated and SATELLITES as an uncoordinated planning problem. The results in Table 1 shows that the plan repairing algorithms are only slightly better (maximally 10%) in terms of the generated communication overhead than replanning, regardless of the number of agents. The trend in Figure 4 shows similar results for various failure probabilities  $P$ . The presented results support the auxiliary hypothesis.

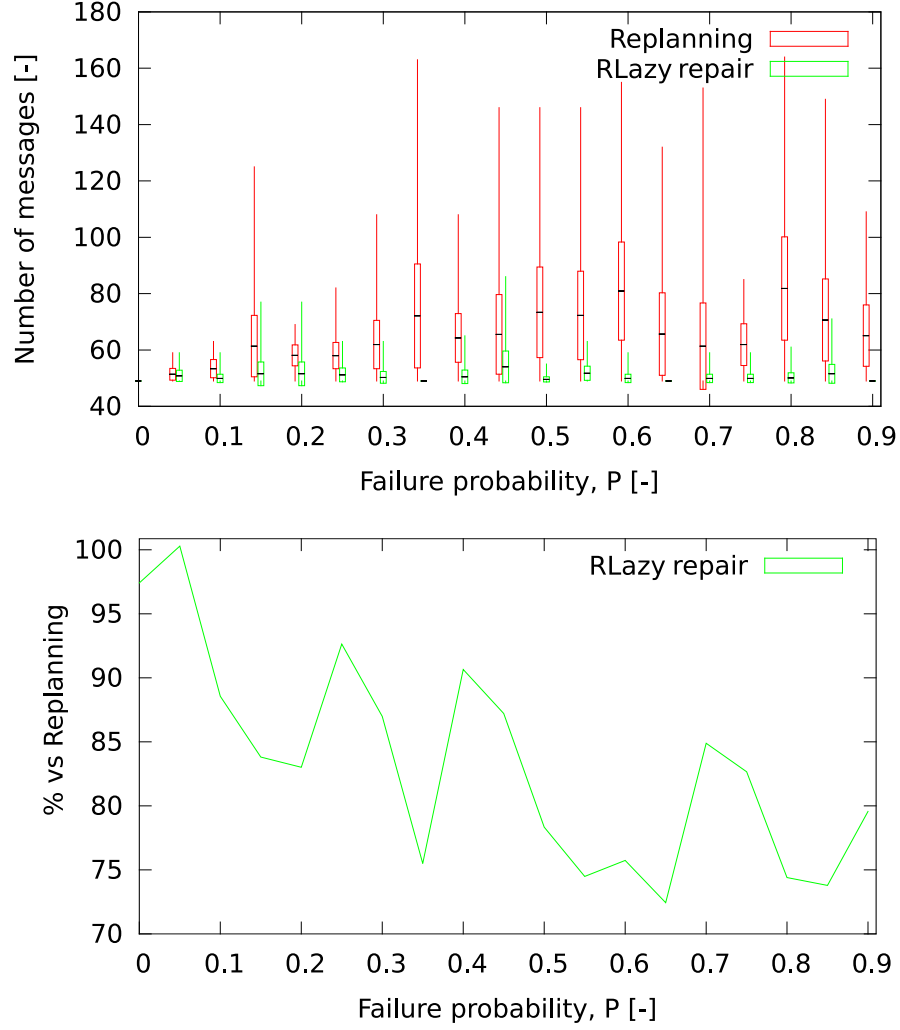


Figure 5: Experimental results for LOGISTICS domain with 3 agents and state perturbations with  $c = 1$ .

The third batch of experiments targeted the perturbation magnitude of the plan failures. The second auxiliary hypothesis we validated states:

*Communication efficiency gain of plan repairing in contrast to replanning should decrease as the difference between the nominal and the corresponding failed states increases.*

The underlying intuition is that, in the case the dynamic environment generates only relatively small state perturbations and the failed states are “not far” from the actual state, the plan repair should perform relatively well. On the other hand, if the state essentially “teleports” the agents to completely different states, replanning tends to generate more efficient solutions than plan repair.

To tackle this hypothesis, we modified the LOGISTICS experiment to simulate state perturbations as the model of the environment dynamics. Figure 5 depicts results of the experiment for  $c = 1$ . The perturbed state for  $c = 1$  is produced by removing one term from the actual state and adding another one. As the chart shows, under random perturbations the plan repairing technique lost its improvement against replanning. For stronger perturbations with  $c = 2, 3, 4$  (not shown in the figure), the ratio between plan repairing and replanning remained on average the same. The trend of the absolute numbers of messages, planning time and execution length was slightly decreasing, as the probability of opportunistic effects increased.

Beside supporting the presented hypotheses the results also show the differences between the two plan repairing algorithms. Table 1 highlights the best results for communication volume and planning time. In most cases the **Repeated-Lazy-Repair** algorithm is more efficient in communication than the **Back-on-Track-Repair** algorithm. The exceptions are the COOPERATIVE PATHFINDING and ROVERS domains with higher numbers of agents. These problems share high combinatorial complexity (COOPERATIVE PATHFINDING in coordination and ROVERS in local planning) and therefore more plan preserving techniques, as **Back-on-Track-Repair**, benefit.

## 6. Final Remarks

In the presented paper, we i) formally introduced the problem of multi-agent plan repair, ii) formulated a notion of relative coordination frequency of a planning problem based on Brafman and Domshlak’s number of coordination points, iii) proposed three algorithms for solving the repair problem and proved their correctness, and finally iv) formulated, as well as experimentally validated the hypothesis stating that under certain conditions, *multi-agent plan repair approaches tend to be more efficient in terms of the communication overhead they generate in comparison to replanning from scratch*. Our results well support the core hypothesis of the paper and we additionally performed a series of experiments validating its boundary conditions articulated

by the series of auxiliary hypotheses in Section 5.

The line of research underlying this paper well correlates with recent works on classical single-agent planning sub-domains, such as partial ordered plan monitoring and repairing [17], conformant [18] and contingency planning [19], plan re-use [3] and plan adaptation [20]. Environment dynamics is also handled by approaches based on Markov decision processes. The main difference to our approach is that the state perturbations utilized in our experiments have *a priori* unknown probabilities. Our own recent approach to the problem of multi-agent plan repair in [21] can be seen only as a precursor to the formal and rigorous treatment of the problem in this paper. Therein, we described the first steps towards a formal treatment of the problem, as well as proposed two specific incomplete algorithms for solving the problem, very distinct from the ones presented here. A sketch of the ideas behind the three plan repairing algorithms were additionally published in our recent work [14], however without a formal description in full details and exhaustive evaluation. Those are presented herein.

There are several open challenges resulting from the presented work. Firstly, the multi-agent planning framework (MA-STRIPS) is not expressive enough to describe certain aspects of concurrent actions and should be extended to this end. This, we suspect, will also influence the multi-agent planning complexity analysis. In particular, there is no way to account for joint actions which have effects strictly different than the union of the individual actions involved. Secondly, there is a need for more efficient and feature-full implementations of multi-agent planners, as the gap between the state-of-the-art classical planners and multi-agent planners is still enormous. Addressing this challenge should also answer the question of scalability potential of the plan repairing approaches based on such planners. Thirdly, there is a lack of standardized planning benchmarks for multi-agent planning, especially considering tightly coordinated planning problems. Exploration of such is needed to further evaluate the hypotheses presented in this paper. Finally, we left out the work towards resolving the validity of Hypothesis 2 aiming at analytical investigation of complexity properties of multi-agent plan repair to future work.

## Acknowledgments

This work was supported by the European Office of Aerospace Research & Development (EOARD) within the grant no. FA8655-12-1-2096, by the

Ministry of Education, Youth and Sports of Czech Republic within the grant no. LD12044, and by the Grant Agency of the Czech Technical University in Prague, grant no. OHK3-060/12.

## References

- [1] R. v. d. Krogt, M. d. Weerdt, Self-interested planning agents using plan repair, in: Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling, 2005, pp. 36–44.
- [2] T. C. Au, H. Munoz-Avila, On the complexity of plan adaptation by derivational analogy in a universal classical planning framework, *Advances in Case-Based Reasoning* (2002) 13–27.
- [3] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Proceedings of ICAPS, 2006, pp. 212–221.
- [4] B. Nebel, J. Koehler, Plan reuse versus plan generation: a theoretical and empirical analysis, *Artificial Intelligence* 76 (1-2) (1995) 427–454.
- [5] J. S. Cox, E. H. Durfee, An efficient algorithm for multiagent plan coordination, in: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05, ACM, New York, NY, USA, 2005, pp. 828–835.
- [6] R. I. Brafman, C. Domshlak, From one to many: Planning for loosely coupled multi-agent systems, in: Proceedings of ICAPS, 2008, pp. 28–35.
- [7] A. Jonsson, M. Rovatsos, Scaling up multiagent planning: A best-response approach, in: F. Bacchus, C. Domshlak, S. Edelkamp, M. Helmert (Eds.), Proceedings of ICAPS, AAAI, 2011.
- [8] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Physics Doklady* 10 (1966) 707.
- [9] R. Dechter, *Constraint processing*, Elsevier Morgan Kaufmann, 2003.
- [10] R. Nissim, R. I. Brafman, C. Domshlak, A general, fully distributed multi-agent planning algorithm, in: Proceedings of AAMAS, 2010, pp. 1323–1330.



- [11] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, *Computational Intelligence* 12 (3) (1993) 268–299.
- [12] R. Zivan, A. Meisels, Asynchronous forward-checking for DisCSPs, *Constraints* 12 (2007) 131–150.
- [13] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, *Journal of Artificial Intelligence Research* 14 (2001) 253–302.
- [14] A. Komenda, P. Novák, M. Pěchouček, Decentralized multi-agent plan repair in dynamic environments (Extended Abstract), in: *Proceedings of AAMAS, 2012*, pp. 1239–1240.
- [15] P. Novák, W. Jamroga, Agents, Actions and Goals in Dynamic Environments, in: T. Walsh (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011*, pp. 313–318.
- [16] IPC, The international planning competition, ICAPS, <http://ipc.informatik.uni-freiburg.de/>.
- [17] C. Muise, S. A. McIlraith, J. C. Beck, Monitoring the execution of partial-order plans via regression, in: *Proceedings of 22nd International Joint Conference on Artificial Intelligence, 2011*, pp. 1975–1982.
- [18] A. Albore, H. Palacios, H. Geffner, Compiling uncertainty away in non-deterministic conformant planning, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), *ECAI, Vol. 215 of Frontiers in Artificial Intelligence and Applications*, IOS Press, 2010, pp. 465–470.
- [19] A. Albore, H. Palacios, H. Geffner, A translation-based approach to contingent planning, in: C. Boutilier (Ed.), *IJCAI, 2009*, pp. 1623–1628.
- [20] H. Muñoz-Avila, M. T. Cox, Case-based plan adaptation: An analysis and review, *IEEE Intelligent Systems* 23 (4) (2008) 75–81.
- [21] A. Komenda, P. Novák, Multi-agent plan repairing, in: *Proceedings of Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities IJCAI-DMPOUW Workshop, 2011*, pp. 1–6.

# Multiagent Plan Repair by Combined Prefix and Suffix Reuse

Antonín Komenda ([komenda@agents.fel.cvut.cz](mailto:komenda@agents.fel.cvut.cz))

Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Peter Novák ([P.Novak@tudelft.nl](mailto:P.Novak@tudelft.nl))

Department of Software and Computer Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands

Michal Pěchouček ([pechoucek@agents.fel.cvut.cz](mailto:pechoucek@agents.fel.cvut.cz))

Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

## Abstract

Deterministic domain-independent multiagent planning is an approach to coordination of cooperative agents with joint goals. Provided that the agents act in an uncertain and dynamic environment, such plans can fail. The straightforward approach to recover from such situations is to compute a new plan from scratch, that is to replan. Even though, in a worst case, plan repair or plan re-use does not yield an advantage over replanning from scratch, there is a sound evidence from practical use that approaches trying to repair the failed original plan can outperform replanning in selected problems. One of the possible plan repairing techniques is based on preservation of fragments of the older plans.

This work theoretically analyses complexity of plan repairing approaches based on preservation of fragments of the original plan and experimentally studies three practical aspects affecting its efficiency in various multiagent settings. We focus both on the computational, as well as the communication efficiency of plan repair in comparison to replanning from scratch and we report on the influence of the following properties on the efficiency of plan repair: (1) the number of involved agents in the plan repairing process, (2) inter-dependencies among the repaired actions, and finally (3) particular modes of re-use of the older plans.

**Keywords:** multiagent systems, automated planning, plan repair

# 1 Introduction

Consider a team of heterogeneous robots working together so as to execute a mission in an environment. Since the robots feature heterogeneous capabilities, it might well be that none of them is able to complete the mission on its own, however by a careful coordination and teamwork, they should be able to reach the joint objective. The team of physical robots is embodied in a dynamic environment in which various events and plan execution interruptions occur and most importantly, in which actions of the agents can fail. To execute their mission, the robots represented by deliberative agents must be able to cope with such a dynamics on both, the individual, as well as the coordination level. Here we focus on the problem of multiagent plan repair which tackles such issue.

Recently, an approach of *multiagent (MA) plan repair* (MA-REPAIR) was proposed in [8], based on multiagent planning (MA-STRIPS) as introduced in [2] and the classical MODELINS principle from [12]. MA-STRIPS is an approach to planning for teamwork and coordination extending the classical STRIPS-based planning techniques. According to the MA-REPAIR approach, the multiagent team computes a team plan using a fully decentralized MA-STRIPS planning algorithm, and subsequently executes the plan, while at the same time monitoring of possible failures of plan execution. Upon an occurrence of such a failure, the team stops execution and invokes a plan repair algorithm and fixes the failed joint plan in order to reach a joint goal state from the state in which the failure occurred.

It can be argued that plan re-use based on MODELINS does not yield much advantage with respect to the computational complexity in the worst case [12], since attempts to fix a failed plan sometimes lead to replanning from scratch anyway. In multiagent and multi-robot settings, where communication is unreliable and costly, however, it is often the communication which is of higher priority than the computational complexity.

In [10], the authors have proposed prefix and suffix-based approaches to multiagent plan repair. These repairing approaches save communication in contrast to replanning from scratch in tightly coupled problems with action failures, however a research question which plan repairing techniques are more appropriate for which planning domains and problems remained unanswered. In this work, we extend our recent experimental study [9], where we have generalized the prefix and suffix-based approaches and present a coherent analysis of computational complexity and practical properties of how particular multiagent plan repair techniques and particular parameterizations perform in different planning domains.

## 2 Multiagent Planning & Repair

We use MA-STRIPS [2] as a model for the multiagent planning problems.

**Definition 1.** Let a quadruple  $\Pi = \langle P, \mathcal{A}, I, G \rangle$  be a *multiagent planning problem* over

- a finite set of propositions  $P$  denoting facts about the environment the agents operate in,

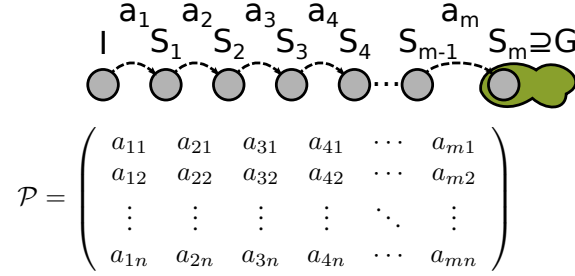


Figure 1: A multiagent plan in matrix form with a visual representation of the intermediate states  $S_1, \dots, S_{m-1}$  and joint actions  $\mathbf{a}_i = (a_{i1}, \dots, a_{in})$ . The gray nodes represent planned states of evolution of the environment,  $I$  is the initial state and  $G$  is the set of goal conditions defining a set of possible goal states. The arcs represent the planned joint actions.

- a set of  $n$  agents  $\mathcal{A} = \{A_1, \dots, A_n\}$ , each characterized by a finite set of actions with STRIPS syntax and semantics the agent can perform, formally  $A_i = \{\langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle \mid \text{pre}(a), \text{add}(a), \text{del}(a) \subseteq P\}$ , where  $\text{pre}(a)$ ,  $\text{add}(a)$ , and  $\text{del}(a)$  represent sets of preconditions, add effects, and delete effects respectively; the transition function describing change of the environment in state  $S$  after execution of action  $a$  into new state  $S'$  is defined as  $S' = (S \cup \text{add}(a)) \setminus \text{del}(a)$  provided that  $\text{pre}(a) \subseteq S$ , i.e., the action  $a$  is *applicable* in the state  $S$ ,
- an initial state  $I \subseteq P$  the environment begin in, and
- a goal state conditions  $G \subseteq P$  characterizing agents' joint objective(s) s.t. a state  $S \subseteq P$  is a goal state iff  $G \subseteq S$ .

Additionally, we define a set of propositions of an action  $a$  as  $\text{prop}(a) = \text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)$  and a set of propositions an agent  $A_i$  affects or is affected by as  $P_i = \bigcup_{a \in A_i} \text{prop}(a)$ . Each action set also contains a empty action  $\epsilon = \langle \emptyset, \emptyset, \emptyset \rangle$ . According to MA-STRIPS, a distinguished subsets of agents' *public* actions known to all other agents is defined as  $A_i^{\text{pub}} = \{a \mid a \in A_i \text{ s.t. } \exists j \neq i : \text{prop}(a) \cap P_j \neq \emptyset\}$  and the complement denoted as *private* actions is defined  $A_i^{\text{priv}} = A_i \setminus A_i^{\text{pub}}$ . A *joint action* of all agents is defined as a  $n$ -tuple  $\mathbf{a} = (a_1, \dots, a_n)$  where for each  $a_i$  holds  $a_i \in A_i$ . Execution of a joint action  $\mathbf{a}$  is defined as  $S' = (S \cup \bigcup_{a \in \mathbf{a}} \text{add}(a)) \setminus \bigcup_{a \in \mathbf{a}} \text{del}(a)$  provided that  $\bigcup_{a \in \mathbf{a}} \text{pre}(a) \subseteq S$  and  $\bigcup_{a \in \mathbf{a}} \text{add}(a) \cap \bigcup_{a \in \mathbf{a}} \text{del}(a) = \emptyset$ .

**Definition 2.** A sequence of joint actions  $\pi = (\mathbf{a}_1, \dots, \mathbf{a}_m)$  is a *multiagent plan* solving a multiagent planning problem  $\Pi = \langle P, \mathcal{A}, S_0, G \rangle$  iff  $\bigcup_{a \in \mathbf{a}_i} \text{pre}(a) \subseteq S_{i-1}$  and  $\bigcup_{a \in \mathbf{a}_i} \text{add}(a) \cap \bigcup_{a \in \mathbf{a}_i} \text{del}(a) = \emptyset$  where for the *intermediate states* hold  $S_i = (S_{i-1} \cup \bigcup_{a \in \mathbf{a}_i} \text{add}(a)) \setminus \bigcup_{a \in \mathbf{a}_i} \text{del}(a)$  for  $i \in 1, \dots, m$  and  $G \subseteq S_m$ .

We will denote the length of a sequence of joint actions  $\pi = (\mathbf{a}_1, \dots, \mathbf{a}_m)$  as  $|\pi| = m$ . A concatenation of two multiagent plans will be denoted as  $(\mathbf{a}_1^1, \dots, \mathbf{a}_k^1) \cdot (\mathbf{a}_1^2, \dots, \mathbf{a}_l^2) = (\mathbf{a}_1^1, \dots, \mathbf{a}_m^1, \mathbf{a}_1^2, \dots, \mathbf{a}_l^2)$ . To index a  $k$ -th joint action in  $\pi$ , we will write  $\pi[k]$  and  $\pi[k, \dots, l]$  where  $1 \leq k \leq l \leq m$  will denote a fragment of the sequence  $(\mathbf{a}_k, \dots, \mathbf{a}_l)$ . A public projection of plan  $\pi$  will be denoted as  $\pi^{\text{pub}}$  and replaces all

actions which are not in any  $a \in A_i^{\text{pub}}$  by the empty action  $\epsilon$ . A visual representation of a multiagent plan is depicted in Figure 1.

With formally defined multiagent planning problems and multiagent plans solving them, we can formally present a problem of multiagent plan repair MA-REPAIR as defined in [10]:

**Definition 3.** Let a quadruple  $\Sigma = \langle \Pi, \pi, F, k \rangle$  be a *multiagent plan repair* problem over

- a multiagent planning problem  $\Pi = \langle P, \mathcal{A}, I, G \rangle$ ,
- an original multiagent plan  $\pi$  solving the problem  $\Pi$  which execution failed s.t. after execution of some  $\mathbf{a}_k$  the resulting state differs from the intermediate state  $S_k$ ,
- a state  $F \subseteq P$  which the system happens to be in, unexpectedly after the plan execution failure, and
- the step  $k \in 1, \dots, |\pi|$ , after which the failure occurred.

A solution to a multiagent plan repair problem  $\Sigma = \langle \Pi, \pi, F, k \rangle$  is a multiagent plan  $\pi'$  solving a modified planning problem  $\Pi' = \langle P, \mathcal{A}, F, G \rangle$ .

Two auxiliary definitions are needed for formal description of the proposed plan repair algorithm.

**Definition 4.** Let forward *proposition propagation* operator  $\oplus$  be a mapping  $\oplus : 2^P \times (\mathbf{A}_1 \times \mathbf{A}_2 \times \dots \times \mathbf{A}_m) \rightarrow 2^P$ , where each  $\mathbf{A} = \times_{i=1}^n A_i$  and  $m \geq 1$ , defined as

$$S \oplus \pi \mapsto \begin{cases} (S \cup \bigcup_{a \in \pi[1] \text{ s.t. } \text{pre}(a) \subseteq S} \text{add}(a)) \setminus \bigcup_{a \in \pi[1] \text{ s.t. } \text{pre}(a) \subseteq S} \text{del}(a) & \text{for } |\pi| = 1, \\ (S \oplus (\pi[1])) \oplus \pi[2, \dots, |\pi|] & \text{otherwise.} \end{cases} \quad (1)$$

Similarly to the transformation operator  $\oplus$ , a reverse-transformation operator is defined as follows:

**Definition 5.** Let *proposition back-propagation* operator  $\ominus$  be a mapping  $\ominus : 2^P \times (\mathbf{A}_1 \times \mathbf{A}_2 \times \dots \times \mathbf{A}_m) \rightarrow 2^P$ , where each  $\mathbf{A} = \times_{i=1}^n A_i$  and  $m \geq 1$ , defined as

$$S \ominus \pi \mapsto \begin{cases} (S \cup \bigcup_{a \in \pi[1]} \text{del}(a)) \setminus \bigcup_{a \in \pi[1]} \text{add}(a) & \text{for } |\pi| = 1, \\ (S \ominus (\pi[|\pi|])) \ominus \pi[1, \dots, |\pi| - 1] & \text{otherwise.} \end{cases} \quad (2)$$

For further use, we define a metrics on actions in a multiagent plan describing importance of the action with respect to the number of actions, which would be no longer applicable if the action is not present in the plan, formally:

**Definition 6.** Let  $A_k^{\text{dep}} \subseteq \mathbf{a}_k$  in a multiagent plan  $\pi$  at step  $k$ . Let a set of actions  $A_{k+1}^{\text{dep}}$  contain all actions of  $\mathbf{a}_{k+1}$  which are dependent on actions of  $A_k^{\text{dep}}$  based on the *effect-precondition relation*, formally  $A_{k+1}^{\text{dep}} = \{a | a \in \mathbf{a}_{k+1} \text{ s.t. } \bigcup_{a' \in A_k^{\text{dep}}} \text{eff}(a') \cap \text{pre}(a) \neq \emptyset\}$ .

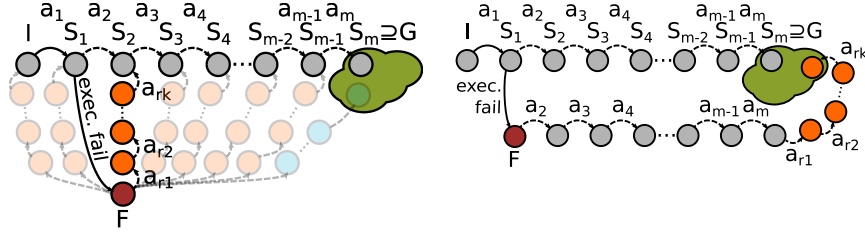


Figure 2: A visual representation of back-on-track (left) and lazy repair (right). The nodes and arcs has the same meaning as in Figure 1. The dashed arcs represent planned but not yet executed actions, the solid ones were already executed. Only the joint action  $a_1$  was executed without a failure. The execution of  $a_2$  failed and the environment ended in the state  $F$ . The orange nodes represent planned states of repair plan  $(a_{r1}, \dots, a_{rk})$ . The pale orange states with their respective actions represent various possibilities of back-on-track repair plans. The cyan nodes represent a solution by replanning. In lazy repair, the application of  $a_2$  and further actions ignores preconditions, which might not be longer satisfied in  $F$ .

The *dependency metrics*  $\text{dep}$  of an action  $a \in \mathbf{a}_k \in \pi$  is then defined as

$$\text{dep}_\pi(a) \equiv \left| \bigcup_{i=k}^{|\pi|} A_i^{\text{dep}} \text{ s.t. } A_k^{\text{dep}} = \{a\} \right|. \quad (3)$$

Besides straightforward multiagent replanning from scratch, which invokes a multiagent planner at the point of a failure and then executes the computed plan right away, two main approaches to multiagent plan repair were presented in [10]: *back-on-track* (BoT) and *lazy repair* (LR).

Both algorithms first formulate a modified multiagent planning problem and rely on the underlying multiagent planner to compute a plan fragment used for re-composition into a solution plan repairing the original failed one. We will refer to the underlying planner as an *inner planner* as it will be used as a component of the plan repair algorithm. The back-on-track strategy (see Figure 2left) tries to fix the prefix of the failed plan by computing a plan from the state in which the system happens to be right after the detection of a plan execution failure to some state along an ideal failure-free execution of the original plan. The resulting multiagent plan re-uses some *suffix* of the original plan, if possible, and extends the plan at its beginning. The idea underlying the lazy repair is complementary (see Figure 2right). Lazy repair takes the remainder of the original plan, re-uses all its actions which still can be executed according to their preconditions regardless of the outcome and completes the plan to some goal state of the planning problem. This way, the resulting plan is composed of re-used *prefix* parts of the original plan with an appended suffix of some new repaired plan. Experiments in [10] showed that these approaches lead to significant savings of communication, as well as computational resources in comparison to replanning from scratch on used planning domains and problems.

Based on the experimental analysis of the two plan repair algorithms and the formal definitions, we can state the core hypotheses of our work here.

Following the theoretical results of Brafman and Domshlak’s complexity analysis of MA-STRIPS planning in [2], our motivation is not to substantially increase the computational complexity of the repair algorithms in respect to the inner planner. Although the communication complexity of the MA-STRIPS planning was not theoretically studied in the literature yet, we cover it in the hypothesis as well, as we hypothesize that the communication complexity is a function of the computational one.

**Hypothesis 1.** *Suffix, prefix or combined multiagent plan repair does not introduce any additional exponential dependency on number of involved agents in respect to the inner planner both in computational and communication complexity.*

Albeit the theoretical results showing that MA-STRIPS planning is not exponentially dependent on the number of involved agent, we hypothesize that practically the overhead of planning or plan repair with higher number of agents grows, especially if the inner planner assumes only public goal condition propositions, which is a common assumption in literature (e.g., in [13]).

**Hypothesis 2.** *Repairing algorithms minimizing the number of agents involved in the plan repairing process tend to generate lower computational and communication overheads than other strategies.*

Immediate repairing of actions, which substantial number of other actions dependent on (see Definition 6), is intuitively more efficient than repair of such actions later with possibly smaller reusable parts of the original plan and higher number of actions to repair in general. Next hypothesis describes this intuition.

**Hypothesis 3.** *Repairing algorithms reusing the original plan as a suffix generate lower computational and communication overheads than the repairing algorithms reusing the original plan as a prefix in domains with actions with high values of the dependency metrics.*

If we parameterize the lengths of reused prefix  $u$  and suffix  $v$  of the plan repairing process, an interesting question is, how do different combinations of these parameters influence the efficiency of the plan repairing process. Let  $m$  be the length of the reusable part of the original plan  $\pi$ , then  $m = |\pi| - k$ , where  $k$  is the step after the failed action. Obviously, for  $u + v < m$ , there will be a gap, which has to be filled by a result of the inner planner, in other words the original plan was *underused*. Reversely, for  $u + v > m$ , there will be an overlap, which has to be reverted, i.e., the original plan was *overused*. Intuitively, these cases are in a sense pathological. The last hypothesis states that repair strategies not underusing nor overusing the original plan should be the most efficient:

**Hypothesis 4.** *Repairing algorithms overusing or underusing the original plan tend to generate higher computational overheads than other algorithms.*

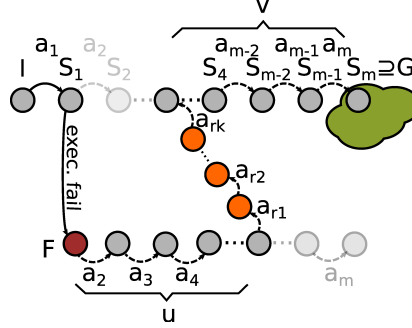


Figure 3: A visual representation of generalized repair. The nodes and arcs have the same meaning as in Figure 2. The repair plan  $(a_{r1}, \dots, a_{rk})$  is used to connect the prefix and suffix of the original plan. The parameters  $u$  and  $v$  prescribe how many actions are reused as the prefix and the suffix respectively.

### 3 Generalized Repair

As outlined in the previous section, the algorithm used in the further analyses is a combination of the lazy and back-on-track approaches, presented in [10], which are orthogonal to each other in how they reuse the original plans. These two approaches can be combined into one algorithm using the original plan both as a prefix and a suffix together. Such an approach *generalizes* the first two approaches and combines the original plan by both fashions as shown in Figure 3.

**Definition 7.** (generalized repair) Let  $\Sigma = \langle \Pi, \pi, F, k \rangle$  be a multiagent plan repair problem and let  $\Pi' = \langle P, \mathcal{A}, F, G \rangle$  be the corresponding modified multiagent replanning problem.

A multiagent plan  $\pi'$  solving  $\Pi'$  is a *generalized repair* of  $\pi$  for vectors of indexes  $U$  and  $V$  iff there is a decomposition of  $\pi'$ , such that  $\pi' = \bar{\pi}_F[k, \dots, k+u] \cdot \pi_{\text{fix}} \cdot \pi[|\pi|-v, \dots, |\pi|]$ , where  $\bar{\pi}_F[k, \dots, k+u]$  is a fragment of plan  $\pi$  omitting inapplicable actions beginning with the state  $F$  for some  $u \in U$ ,  $\pi_{\text{fix}}$  is a new plan connecting the reused fragments, and  $\pi[|\pi|-v, \dots, |\pi|]$  is a fragment of the original plan for some  $v \in V$ . The vectors contain only valid indexes to the reused part of the multiagent plan  $\pi$ , that is  $\forall u \in U : 0 \leq u \leq |\pi| - k$  and  $\forall v \in V : 0 \leq v \leq |\pi| - k$ , to meet the requirements of the decomposition parts  $\bar{\pi}_F[k, \dots, k+u]$  and  $\pi[|\pi|-v, \dots, |\pi|]$ . It holds that  $|U| = |V|$  and for  $\forall i$  s.t.  $1 \leq i \leq |\pi|$  there is no  $j : 1 \leq j \leq |\pi|, j \neq i$  s.t.  $(u_i, v_i) = (u_j, v_j)$ . Also  $0 \in U, 0 \in V$ .

To illustrate the generalized notion of this repair, it can be shown that for  $U = (|\pi| - k), V = (0)$  the approach becomes the lazy repair and for  $U = (0), V = (|\pi| - k, \dots, 0)$  the back-on-track approach. In the first case, the original plan is reused as a plan fragment ignoring preconditions of length  $|\pi| - k$  equally to the definition of lazy repair in [10]. In the other case, the definition of the index vector  $V$  implies trying to reuse a plan fragment starting with length  $|\pi| - k$  and ending with length 0, equally to the back-on-track repair definition in [10]. Finally,  $U = (0), V = (0)$  describes



replanning.

It could be argued that generalization reusing the original plan only as prefix and suffix parts is in fact not the only possible repair scheme, *e.g.*, by means of the MODDELINS scheme presented in [12]. The MODDELINS reuse scheme describing the presented generalized repair approach is

$$(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_u) \cdot \pi_{\text{fix}} \cdot (\mathbf{a}_{|\pi|-v}, \mathbf{a}_{|\pi|-(v-1)}, \dots, \mathbf{a}_{|\pi|}), \quad (4)$$

however a scheme

$$(\mathbf{a}_1, \dots, \mathbf{a}_u), \pi_{\text{fix}}^1, (\mathbf{a}_{u+1}, \dots, \mathbf{a}_{u+v}), \pi_{\text{fix}}^2, (\mathbf{a}_{|\pi|-w}, \dots, \mathbf{a}_{|\pi|}) \quad (5)$$

would be also possible, but generalized repair cannot directly represent it. Since the motivation of the generalized repair is not to be general in the sense of reuse pattern of the original plan, but be general from perspective of reuse of the original plan as both prefix and suffix plan fragments. For such case, the generalized repair scheme is complete, since breaking the  $\pi_F[k, \dots, k+u]$  or the  $\pi[|\pi|-v, \dots, |\pi|]$  fragments into smaller parts would require additional  $\pi_{\text{fix}}$  plans, which could be concatenated into one central fixing fragment as defined in Definition 7.

The algorithm for the generalized repair is outlined in Algorithm 1. In the case, a failure is detected by the agent team, the current state after the failure is retrieved and the plan repair algorithm for the plan repair problem  $\Sigma = \langle \Pi, \pi, F, k \rangle$  is invoked. In each plan repair attempt a modified multiagent planning problem is formulated according to the current values of  $u$  and  $v$  prescribing the length of the reused prefix and suffix of the original plan. These parameters are took from two vectors  $U$  and  $V$  parameterizing the repair.

If a repair plan is found, the repair process finishes, otherwise another attempt with a different combination of  $u$  and  $v$  is made (selection of  $u$  has priority over  $v$  in the combination of indexes). The resulting repairing plan consists of three components: the preserved prefix of the original plan  $\pi_{\text{pre}}$ , a newly computed infix  $\pi_{\text{fix}}$  and suffix part  $\pi_{\text{suf}}$ , again preserving a part of the original plan  $\pi$ .

The preserved prefix part  $\pi_F$  of the original plan corresponds to a plan fragment of  $\pi$  ignoring the preconditions such that only actions applicable in sequence beginning from state  $F$  are used. The actions with unmet preconditions are simply omitted. Additionally, the prefix  $\pi_{\text{pre}}$  is based only on a part of the original plan effectively reusing  $u$  actions beginning after the  $k$ -th action of the original plan  $\pi$ . The suffix part  $\pi_{\text{suf}}$  is obtained as the last  $v$  actions of the original plan  $\pi$ .

Finally, the infix part of the plan is computed by invocation of the inner multiagent planner MA-Plan<sup>1</sup>. The initial state of the modified planning problem is the state in which a failure-free execution of the repair prefix  $\pi_{\text{pre}}$  would result in starting from the state  $F$ , that is propagation  $F \oplus \pi_{\text{pre}}$ . The set of goal states  $G \ominus \pi_{\text{suf}}$  corresponds to a back-propagation of effects of the preserved suffix component  $\pi_{\text{suf}}$  from the set of original goals  $G$ .

If the multiagent planner finds a plan for the modified planning problem, the repair plan takes the form  $\pi_{\text{pre}} \cdot \pi_{\text{fix}} \cdot \pi_{\text{suf}}$  and gets executed from the failure point on. In

<sup>1</sup>In the experiments, we used the Planning First implementation of a MA-STRIPS planner from [13].

---

**Algorithm 1** Generalized-Repair( $\Sigma, U, V, \text{MA-Plan}$ )

---

**Input:** A multiagent plan repair problem  $\Sigma = \langle \Pi, \pi, F, k \rangle$ .

**Input:** Parameters  $U$  and  $V$  prescribing the lengths for reusing of the original plan as prefix and suffix respectively.

```
1:  $u, v$  = initial pair of  $u \in U$  and  $v \in V$ 
2: repeat
3:    $\pi_{\text{pre}} = \bar{\pi}_F[k, \dots, k + u]$ 
4:    $\pi_{\text{suf}} = \pi[|\pi| - v, \dots, |\pi|]$ 
5:    $\pi_{\text{fix}} = \text{MA-Plan}(\langle P, \mathcal{A}, F \oplus \pi_{\text{pre}}, G \ominus \pi_{\text{suf}} \rangle)$ 
6:   if  $\pi_{\text{fix}} \neq \emptyset$  then
7:      $\pi = \pi_{\text{pre}} \cdot \pi_{\text{fix}} \cdot \pi_{\text{suf}}$ 
8:     break
9:   end if
10: until tested all pairs of  $u \in U$  and  $v \in V$ 
11: if  $\pi = \emptyset$  then return fail
```

---

the case no repair plan can be found, the algorithm attempts the repair for a different combination of  $u$  and  $v$  until either a repair plan is found, or it turns out that no repair for the failure exists.

The description of the algorithm will be concluded with proofs of soundness and completeness. Since the generalized repair algorithm uses inner invocation of the inner multiagent planner similarly to back-on-track and lazy repairs [10], its correctness relies on the correctness of the inner planner.

**Lemma 8.** (*soundness*). Let  $\Pi = \langle P, \mathcal{A}, I, G \rangle$ , be a multiagent planning problem with agents situated in a dynamic environment in which the environment can interfere with the plan execution and let  $\pi$  be a solution to  $\Pi$ . Let also  $F$  be a state resulting from an interference of the environment, a plan failure, at a step  $k$  of execution of the plan  $\pi$ .  $\Sigma = \langle \Pi, \pi, F, k \rangle$  denotes the corresponding multiagent plan repair problem.

Provided that the execution of **Generalized-Repair**( $\Sigma, U, V, \text{MA-Plan}$ ) does not fail, but finishes with a resulting plan  $\pi'$  and **MA-Plan** is a sound MA-STRIPS planner, a failure-free execution of  $\pi'$  leads to some goal state of the original multiagent planning problem  $\Pi$ .

*Proof.* Regardless what particular state  $F \oplus \pi_{\text{pre}}$  the failure-free execution of the applicable actions from the fragment  $\bar{\pi}_F$  ends up in, the solution plan  $\pi_{\text{fix}}$ , if exists, to the problem  $\langle P, \mathcal{A}, F \oplus \pi_{\text{pre}}, G \ominus \pi_{\text{suf}} \rangle$  will take the system from the state  $F \oplus \pi_{\text{pre}}$  to a state  $G \ominus \pi_{\text{suf}}$  corresponding to the original multiagent planning problem  $\Pi$ . The back-propagated propositions  $G \ominus \pi_{\text{suf}}$  either represent a required (part of) state along the original execution trace of the original plan  $\pi$  and then the remainder  $\pi_{\text{suf}}$  leading to an original final state is reused, or a failure-free execution of  $\pi_{\text{fix}}$  leads directly to one of the final states defined by  $G$  without reusing a part of  $\pi$  as suffix  $\pi_{\text{suf}} = \emptyset$ , and therefore  $G \ominus \pi_{\text{suf}} = G$ .  $\square$

The initial part and the final part of the proof is based on the soundness proofs of lazy repair and back-on-track respectively in [10]. As mentioned before, in general-

ized repair, these two approaches merge, therefore the proofs are based on the same argumentation.

**Lemma 9.** (completeness). *Let  $\Pi = \langle P, \mathcal{A}, I, G \rangle$  be a multiagent planning problem and let  $\pi, F, k$ , as well as  $\Sigma$  are as in the Lemma 8. Let  $U$  and  $V$  be integer vectors by Definition 7.*

*If there exists a solution plan to the multiagent planning problem  $\Pi' = \langle P, \mathcal{A}, F, G \rangle$  and **MA-Plan** is a complete MA-STRIPS planner, then the execution of **Generalized-Repair**( $\Sigma, U, V, \text{MA-Plan}$ ) finishes and finds a repair plan of  $\pi$ .*

*Proof.* The algorithm tests all combinations of  $U$  and  $V$  values. It eventually tests the required combination  $u = 0$  and  $v = 0$ . Based on the definition of the algorithm, in such case,  $\pi = \pi_{\text{pre}} \cdot \pi_{\text{fix}} \cdot \pi_{\text{suf}}$  degenerates to  $\pi = \pi_{\text{fix}}$  since  $\bar{\pi}_F[k, \dots, k + u] = \bar{\pi}_F[k, \dots, k] = \emptyset$  and  $\pi[|\pi| - v, \dots, |\pi|] = \pi[|\pi|, \dots, |\pi|] = \emptyset$ . The  $\pi_{\text{fix}}$  is then a solution of  $\Pi'$  generated by **MA-Plan**. If such solution exists, it is found, since **MA-Plan** is assumed to be complete.  $\square$

The principle of the completeness proof follows the idea of the back-on-track proof and in addition it requires fewer assumptions than the lazy repair which is complete only for dead-end free planning problems.

## 4 Complexity Analysis

In this section, the presented plan repair algorithm will be theoretically studied from perspective of a classical complexity metrics and one additional metrics suitable for distributed algorithms. The classically studied metrics is time complexity. Additionally, in multiagent systems, one can use a metrics based on an asymptotic ratio of communication volume required for an algorithm to finish to size of the input, similarly as in the case of the time complexity.

### 4.1 Time Complexity of MA-Plan

The generalized plan repair algorithm presented in the previous section use a multiagent planner as a component, therefore its complexity is an key part of the further analysis. The time complexity of the multiagent planning based on solving of coordination Constraint Satisfaction Problem (CSP) and internal heuristic search (and therefore the **MA-Plan** implementation of the planner presented in [13] as Planning First) was studied in [2], therefore the analysis of the time complexity from [2] will be recalled in the following paragraphs.

Informally, the complexity is “the number of times [needed] to verify that a certain choice of coordination-sequence length forms a basis for a solution  $\times$  the complexity of the verification process”. To formally describe the time complexity of the **MA-Plan** approach, firstly [2] define size of the CSP domains for each agents’ CSP variable. Each value of each domain represents one possible coordination sequence for one agent. Such sequences consist of at most  $\delta$  coordination points defined as pairs  $(a, t)$  with a public action  $a$  and  $1 \leq t \leq n\delta$  for  $n$  agents.

The idea of the  $n\delta$  limit for the virtual time points  $t$  can be demonstrated on an example with  $n = 2$  agents  $A_i, A_j, i \neq j$  and  $\delta = 3$  with precisely three used coordination points for both agents:

$$\begin{aligned} A_i : & \begin{pmatrix} a_1^i & * & a_2^i & * & a_3^i & * \end{pmatrix} \\ A_j : & \begin{pmatrix} * & a_1^j & * & a_2^j & * & a_3^j \end{pmatrix}. \end{aligned} \quad (6)$$

The example shows the longest possible coordination pattern for that particular instance as both the agents use all coordination points possible and the actions depends on each other such that no prolonging of the pattern is possible. In that case, the first coordination point is  $(a_1^i, 1)$  and the last one  $(a_3^j, n\delta)$ , where  $n\delta = 6$ .

The size of the CSP domain as defined in [2] is for an agent  $A_i$

$$|D_i| = \sum_{d=1}^{\delta} \binom{n\delta}{d} \cdot |A_i^{\text{pub}}|^d = O((n\delta|A_i^{\text{pub}}|)^{\delta+1}). \quad (7)$$

The term  $\binom{n\delta}{d}$  represents all possible combinations of  $d$  virtual time points for the public actions (e.g., for  $d = 2, n\delta = 6$  there are 15 of them  $\{(1, 2), (1, 3), \dots, (1, 6), (2, 3), (2, 4), \dots, (5, 6)\}$ ) and the term  $|A_i^{\text{pub}}|^d$  represents all possible public action sequences of length  $d$  of agent  $\alpha$  (e.g., for  $d = 2$  and  $|A_i^{\text{pub}}| = 2$  and the sequences are  $\{a_1 a_1, a_1 a_2, a_2 a_1, a_2 a_2\}$ ), therefore for each  $d$ , the complete term in the sum counts the number of possible coordination sequences for  $d$  coordination points. Finally, the summed up result represent the number of all possible coordination sequences for one agent.

The domain size is then used in the final time complexity formula for the *internal planning constraints* (ipc) in the CSP in the following form

$$O(f(\mathcal{I}) \cdot n \cdot \max_{A_i \in \mathcal{A}} |D_i|) = O(f(\mathcal{I}) \cdot n(n\delta|A^{\text{pub}}|)^{\delta+1}) = O_{\text{ipc}}, \quad (8)$$

where the term  $f(\mathcal{I})$  represents maximal complexity of individual planning  $\mathcal{I}$  with a function  $f$  describing the cost of switching from regular planning and  $A^{\text{pub}} = \bigcup_{i=1}^n A_i^{\text{pub}}$ .

The complexity induced by the *coordination constraints* (cc) is in [2] derived from time complexity of **Adaptive-Tree-Consistency** algorithm (ATC) for solving CSP problems. The complexity is based on a tree-width  $\omega$  of the CSP constraint graph [5] which is

$$O(n \cdot \max_{A_i \in \mathcal{A}} |D_i|^{\omega+1}) = O(n(n\delta|A^{\text{pub}}|)^{\delta\omega+\epsilon}) = O_{\text{cc}}, \quad (9)$$

where  $\epsilon = \delta + \omega + 1$  is dominated by  $\delta\omega$ . According to [2], the constraint graph is isomorphic to the moral graph of agent interaction graph, therefore  $\omega$  can be treated as a tree-width of the agent interaction graph. An *agent interaction graph* describes dependencies of the agents on each other defined by public actions.

The final complexity is a sum of the complexities from Eq. 8 and Eq. 9 for the particular constraints

$$O_{\text{ipc}} + O_{\text{cc}} = O(f(\mathcal{I}) \cdot n(n\delta|A^{\text{pub}}|)^{\delta+1} + n(n\delta|A^{\text{pub}}|)^{\delta\omega+\epsilon}) = O_{\text{MAP}}. \quad (10)$$

The complexity has no direct exponential dependence on the number of agents  $n$ , has no direct exponential dependence on the length of the individual plans of the agents and has no direct exponential dependence on the size of the original planning problem  $|\Pi|$ . However, the complexity of the individual planning  $f(\mathcal{I})$  is in general still exponential in the size of the individual planning problems.

## 4.2 Time Complexity of the Generalized Repair Algorithm

Let  $\Pi = \langle P, \mathcal{A}, I, G \rangle$  be the original multiagent planning problem and  $\Pi' = \langle P, \mathcal{A}, F, G \rangle$  be the related multiagent replanning problem. The time complexity of the planning problem  $\Pi$  is complexity of the MA-Plan algorithm as showed in the previous section

$$O(f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + n(n\delta q)^{\delta\omega+\epsilon}), \quad (11)$$

where for brevity  $q = |A^{pub}|$ . Straightforwardly, the replanning complexity is in general the same as of planning, since the only difference is another initial state  $F$ . That means  $n, q$  and  $\omega$  are the same<sup>2</sup>. The length  $\delta$  depends on the particular initial state, however there is no guarantee that  $\delta$  for  $F$  will be generally higher or lower than  $\delta$  for  $I$ . From [12], it is known that plan reuse cannot be generally less complex than replanning from scratch.

**Lemma 10.** (time complexity). *Let  $\Sigma = \langle \Pi, \pi, F, k \rangle$  be a multiagent plan repair problem and let  $\Pi = \langle P, \mathcal{A}, I, G \rangle$  be the related multiagent planning problem. Let  $U$  and  $V$  be the index vectors of generalized plan repair by Definition 7.*

*The asymptotic time complexity of Generalized-Repair( $\Sigma, U, V, \text{MA-Plan}$ )*

$$O_{GEN} = O(f(\mathcal{I}) \cdot l^2 n(n\delta q)^{\delta+1} + l^2 n(n\delta q)^{\delta\omega+\epsilon} + 2l^3 \mathcal{G}^2 + 2l), \quad (12)$$

where  $\mathcal{G} = |\Pi|$ ,  $l = |\pi|$ ,  $q = |A^{pub}|$ , and the rest of the symbols follow the definitions for Eq. 10 from [2].

*Proof.* The Generalized-Repair algorithm is parametrized by two index vectors  $U$  and  $V$  which are used in a repeated search for a solution of the multiagent plan repair problem. The time complexity is informally *how many times the algorithm needs generate and test a repair strategy*  $\times$  *what is the complexity of the generate and test procedure*.

The generate and test procedure consists of two proposition propagation procedures (by Definitions 4 and 5) each in worst case using simulation of time complexity, which in the worst case require  $l$  testings of all actions'  $|A|$  possible preconditions  $|P|$ , therefore  $O(l|A||P|) = O(l\mathcal{G}^2)$ , since the number of actions and propositions cannot be bigger than the size of the whole planning problem. Technically, the extraction of the prefix fragment  $\pi_{pre}$  is part of the proposition propagation process  $F \oplus \pi_{pre}$ , therefore there is one  $O(l\mathcal{G}^2)$  term. Another  $O(l\mathcal{G}^2)$  term is needed for the proposition back-propagation  $G \ominus \pi_{suf}$  similarly related to extraction of the suffix  $\pi_{suf}$ . The  $\pi_{fix}$  fragments requires solving one multiagent planning problem, hence the time complexity of one generate and test procedure is

$$O_{GT} = O_{MAP} + O(2l\mathcal{G}^2) = O(f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + n(n\delta q)^{\delta\omega+\epsilon} + 2l\mathcal{G}^2). \quad (13)$$

<sup>2</sup>The difference in sizes of different states cannot be larger than  $|P|$ , which bounds it by a constant.

The procedure with complexity  $O_{GT}$  is in the **Generalized-Repair** used maximally  $|U| \cdot |V|$  times. If a solution is found, two additional concatenations are needed to finally build the solution, therefore

$$\begin{aligned} O(|U| \cdot |V|) \cdot O_{GT} + O(|\pi_{\text{pre}}|) + O(|\pi_{\text{suf}}|) &= \\ O(l^2 \cdot f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + l^2 \cdot n(n\delta q)^{\delta\omega+\epsilon} + 2l^3\mathcal{G}^2 + |\pi_{\text{pre}}| + |\pi_{\text{suf}}|) &= \\ O(f(\mathcal{I}) \cdot l^2 n(n\delta q)^{\delta+1} + l^2 n(n\delta q)^{\delta\omega+\epsilon} + 2l^3\mathcal{G}^2 + 2l) &= O_G(14) \end{aligned}$$

where  $|U| \cdot |V| = O(|\pi|^2) = O(l^2)$ , as the index vectors can parametrize at most all combinations of the indices to the original plan  $\pi$  by Definition 7. The lengths of the resulting plan segments  $|\pi_{\text{pre}}|$  and  $|\pi_{\text{suf}}|$  are bounded by the length of the plan  $l = |\pi|$ .  $\square$

The resulting time complexity of generalized repair algorithm does not comprise any extra exponential dependency on any of the additional terms, which are always polynomial provided that  $l$  is polynomial w.r.t.  $\mathcal{G}$ . Consistently with [12], the asymptotic worst-case complexity is also never reduced, which is anticipated result of the analysis. The idea of the presented plan repair technique in general is of lowering  $\delta$  by simplifying the inner planning process with help of reuse of parts of the original plan. Since  $\delta$  is in  $O_{MAP}$  in two exponent terms, such idea is *positively supported by the analysis* as well.

The results of the time complexity analysis of the repair algorithms therefore *prove* the time complexity part of the first stated Hypothesis 1 with an additional assumption on the polynomial length of the repaired plan. Additionally, they are not in conflict with the remaining three hypotheses. Hypothesis 2 targets taking only a subset of  $\mathcal{A}$  which can in effect lower the tree-width  $\omega$  if the remaining agents are less coupled. In Hypothesis 3, the length  $\delta$  of the inner repair plan is targeted, as in the cases of problems with actions having long dependency trees, it is theorized that fixing the problem sooner will require smaller  $\delta$  than solving it later possibly with longer reverting plan of a bigger  $\delta$ . In the last Hypothesis 4, smaller  $\delta$  should be achieved by possibly short repair plans, where no reverting is caused by overusing of the original plans  $u \in U, v \in V, u + v > l - k$  and no unnecessarily long repair plans are needed provided that  $u + v < l - k$ .

### 4.3 Communication Complexity of MA-Plan

For the study of communication complexity of the presented multiagent plan repair algorithm, firstly, the communication complexity of the inner planning process has to be known. Since the work of Brafman and Domshlak [2] formally tackle only time complexity, we propose a analysis of communication in a similar manner based on an analysis of the **Adaptive-Tree-Consistency (ATC)** algorithm.

The communication complexity of ATC can be derived from space complexity which was studied in [5]. The analysis will use the Big- $O$  notation similarly to the previous section. To distinguish time and communication complexity, the Big- $O$  will use superscript  $c$  for communication complexity  $O^c$ .

Size of each message in ATC is  $\max_{A_i \in \mathcal{A}} |D_i|^{\text{sep}}$ , where sep is size of a maximal separator size in tree decomposition of the CSP. The size of the separator is in bucket-trees [5]

the tree-width  $\omega$ , therefore a worst case size of one message is  $\max_{A_i \in \mathcal{A}} |D_i|^\omega$ . Maximal number of messages communicated in a bucket-tree CSP solver is double the number of arcs (two messages for each arc in tree of  $n$  vertex graph), therefore  $2(n-1)$ , where  $n$  is the number of the buckets. The buckets represent the CSP variables (with constraints in them resembling the principle of tree-decomposition), therefore the number of the buckets is the number of agents in the coordination constraint (cc). Since the internal planning constraints (ipc) are represented as unary constraints, they do not require any communication. All together, it gives the communication complexity of the MA-STRIPS planning as

$$2(n-1) \cdot \max_{A_i \in \mathcal{A}} |D_i|^\omega = O^c(\varepsilon n(n\delta q)^{\delta\omega+\epsilon} + \epsilon') = O^c(n(n\delta q)^{\delta\omega+\epsilon}) = O_{MAP}^c, \quad (15)$$

where  $\epsilon$  is dominated by  $\delta\omega$  in the exponent,  $\varepsilon = 2$  is a polynomial coefficient and  $\epsilon'$  is dominated by the first polynomial term. The communication complexity of planning using CSP for coordination is therefore not exponentially dependent on the number of agents  $n$ , it is not dependent on the complexity of the individual planning  $\mathcal{I}$  and it has no direct exponential dependence on the size of the original planning problem  $|\Pi|$ , similarly to the time complexity. The communication complexity is bounded by one exponential term in the number of the coordination points and tree-width of the agent interaction graph

$$\exp(\delta\omega). \quad (16)$$

#### 4.4 Communication Complexity of the Generalized Repair Algorithm

The communication complexity of generalized repair will be derived by an equal process as in the case of the time complexity. It builds on the derived complexity of the inner planning of MA-Plan which is in the case of communication  $O^c(n(n\delta q)^{\delta\omega+\epsilon})$  as showed in Eq. 15.

Equally to the time complexity, replanning is in general the same as planning from the perspective of communication complexity. The only difference is in the initial state. That means  $n, q$  and  $\omega$  are the same and  $\delta$  depends on the particular initial state without any general guarantees on its change during replanning.

**Lemma 11.** (*communication complexity*) Let  $\Sigma = \langle \Pi, \pi, F, k \rangle$  be a multiagent plan repair problem and let  $\Pi = \langle P, \mathcal{A}, I, G \rangle$  be the related multiagent planning problem. Let  $U$  and  $V$  be the index vectors of generalized plan repair by Definition 7.

The asymptotic communication complexity of **Generalized-Repair**( $\Sigma, U, V, \text{MA-Plan}$ )

$$O_{GEN}^c = O^c(l^2 n(n\delta q)^{\delta\omega+\epsilon} + 2nl^3 \mathcal{G} + n^2 + n), \quad (17)$$

where  $\mathcal{G} = |\Pi|$ ,  $l = |\pi|$ ,  $q = |A^{pub}|$ , and the rest of the symbols follow the definitions for Eq. 10 from [2].

*Proof.* In the **Generalized-Repair**, the process is separable to repeated sub-processes of generating and testing of a repair strategy and it needs an additional synchronization

broadcast as the agents has to be aware of new plan repair process in case of failure in a private fact. Such broadcast is an additive factor  $O_{\text{sync}}^c = O(n)$  as the messages are sent to all agents.. The communication complexity will be firstly derived for one such sub-process. The two proposition propagation procedures  $F \oplus \pi_{\text{pre}}$  and  $G \ominus \pi_{\text{suf}}$ , each in the worst case use the same principle as a distributed simulation of execution of a multiagent plan segment  $O(nl|P|) = O(nl\mathcal{G})$  and are used with one inner planning, therefore

$$O_{GT}^c = O^c(n(n\delta q)^{\delta\omega+\epsilon} + 2nl\mathcal{G}). \quad (18)$$

Equally to the time complexity, the sub-process with the complexity  $O_{GT}^c$  can be used in worst case  $|U| \cdot |V|$  times. If a sound solution is found the agents has to inform each other, therefore there is beside the initial synchronization a termination synchronization of  $O^c(n^2)$ , although the local plans has not to be communicated as they are later executed by the particular agents. The communication complexity of the **Generalized-Repair** is

$$\begin{aligned} O_{\text{sync}}^c + O^c(|U| \cdot |V|) \cdot O_{GT}^c + O^c(n^2) &= \\ O^c(n + l^2 n(n\delta q)^{\delta\omega+\epsilon} + 2nl^3\mathcal{G} + n^2) &= \\ O^c(l^2 n(n\delta q)^{\delta\omega+\epsilon} + 2nl^3\mathcal{G} + n^2 + n) &= O_{GEN}^c, \end{aligned} \quad (19)$$

where  $|U| \cdot |V| = O(|\pi|^2) = O(l^2)$ , as the index vectors can parametrize at most all combinations of indices to the original plan  $\pi$  by Definition 7.  $\square$

The analyzed communication complexity do not bring any new terms exponentially dependent on any of the parameters, which are always polynomial provided that  $l$  is polynomial w.r.t.  $\mathcal{G}$ . Therefore the communication complexity of the proposed plan repair algorithm remains exponential only in the factor of number of coordination points  $\delta$  in the inner repair plan and tree-width  $\omega$  of the agent interaction graph, i.e.,  $\exp(\delta\omega)$  as in Eq. 16. This result is anticipated as the communication complexity is usually proportional to the time complexity.

The resulting communication complexity of the generalized repair algorithm *proves* the communication and final part of Hypothesis 1 with an additional assumption on the polynomial length of the repaired plan. Additionally, the results support the experimental results from [10] and concur with the remaining hypotheses, similarly as in the case of the time complexity. The core hypothesis of [10] states that the communication overhead is lowered by plan repair producing more preserving repairs in comparison to replanning. Since the communication complexity of replanning is exponentially dependent on  $\delta$  this hypothesis is supported by the analysis as far as at least one coordination point is spared, because decreasing the exponential factor by one  $\exp((\delta - 1)\omega)$  dominates any additional polynomial factors added by the plan repair techniques. This is true only, if the problems are tightly coordinated  $\omega \gg 0$ . If it is the contrary, the exponential factor is negligible even if  $\delta$  is not decreased by the preservation of the repair, formally  $\exp(\delta\omega) \rightarrow 1$  iff  $\delta \rightarrow 0$  or  $\omega \rightarrow 0$ .

The arguments on decreasing the amount of communication used for the remaining three hypothesis of this article copies those in the time complexity analysis. Hypothesis 2 targets taking only a subset of  $\mathcal{A}$ , which can in effect lower the tree-width  $\omega$  if the



---

**Algorithm 2** Plan execution and monitoring scheme.

**Input:** An initial multiagent planning problem  $\Pi = \langle P, \mathcal{A}, I, G \rangle$ , vectors  $U$  and  $V$ , and Planning First multiagent planner by [13] as MA-Plan.

```
1:  $\pi = \text{MA-Plan}(\Pi)$ 
2: if MA-Plan failed then return fail
3:  $k = 1$ 
4:
5: repeat
6:   agents perform  $\pi[k]$ 
7:   if failure detected then
8:     retrieve the current state  $F$  from the environment
9:      $\pi = \text{Generalized-Repair}(\langle \Pi, \pi, F, k \rangle, F, G, \text{MA-Plan})$ 
10:     $k = 1$ 
11:   else
12:      $k = k + 1$ 
13:   end if
14: until Generalized-Repair failed or  $k > |\pi|$ 
```

---

remaining agents are less coupled, and therefore lower the communication complexity. In Hypothesis 3, the length  $\delta$  of the inner repair plan should be minimized if failures of actions with long dependency trees are fixed as soon as possible. In Hypothesis 4, smaller  $\delta$  should be achieved by possibly short repair plans by appropriate reusing of the original plan.

## 5 Experimental Analysis

The further sections of the article focus on the remaining hypotheses and its experimental analysis.

The experiments were conducted in a synthetic setting, a simulated world with a group of agents using a plan execution, monitoring and repair loop (see Algorithm 2). The world was modeled as fully observable. All failures of plan execution were generated by the simulator according to a uniform distribution over time and parametrized by a probability  $p$  of failure occurrence in each step for each experiment. The failures were handled by the agents immediately upon detection.

A failure was simulated by not-execution of some of the agent actions from the actual plan step. The particular actions were chosen according to an uniform probability distribution over the individual actions within a joint action. As showed by [10], failure models with more radical impacts on the environment (e.g., state perturbations) decrease usability of the plan repairing approaches. Our motivation in this work is to study types of plan repairing strategies, therefore we stick only to action failures.

For the implementation of the experimental setup and the repairing algorithms, we used a centralized world simulator integrating the multiagent domain-independent planner Planning First [13] denoted as MA-Plan. Each agent run in its own thread

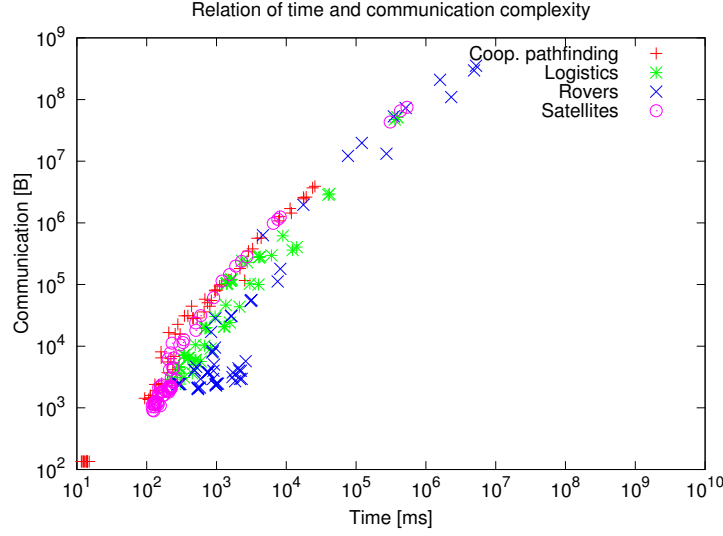


Figure 4: Relation between communicated bytes and computation time for solving the plan repairing problems.

and deliberated asynchronously. The experiments were executed on 8-core processor at 3.6GHz with Java Virtual Machine limited to 2.5GB of RAM.

For the experiments, we used four planning domains. Three of them originate in the standard single-agent IPC planning benchmarks. Similarly to the evaluation of the MA-Plan algorithm in [13], we chose domains, which are straightforwardly modifiable to a multiagent setting: LOGISTICS, ROVERS, and SATELLITES. Additionally, we have extended the set of benchmarks by COOPERATIVE PATHFINDING coordination domain on a grid [10].

The experimental measurements were based on two metrics focusing on the target efficiencies: cumulative time consumed by the particular plan repairing algorithms during a single run of the simulation, i.e., the overall time spent in the algorithm (incl. the underlying planning process) excluding the initial planning phase of the scheme (Algorithm 1). The second metric was communication complexity of the process, that is the volume of communicated information in bytes among the involved agents during the plan repairing processes. Those are mainly the messages generated by the DisCSP solver of the Planning First MA-Plan planner and an additional synchronization processes minimizing the number of agents involved in the plan repairing process.

To account for differences in essential computational and communication complexity of the domains, we conducted a relationship experiment between these two measures. Figure 4 depicts the results and demonstrates that there is no essential discrepancy between the computational and communication complexity of the plan repairing solutions. That means the following results are not biased by problems extremely hard in time and simple in communication and vice versa.

## 5.1 The Number of Repairing Agents

Regardless of the theoretical results presented in [2], showing that the computational complexity of CSP-based multiagent planning is not exponentially dependent on the number of the agents, in practical experiments, we faced a non-negligible dependence of this number and required communication and computational effort. The first set of experiments analyzes this relation by means of Hypothesis 2.

### 5.1.1 Used Plan Repair

To validate Hypothesis 2, we have prepared an extensive set of plan repairing strategies stemming from the generalized repair. They can be divided into three main groups: one *without agent* count minimization, and two *with agent minimization*. First of the minimization groups reuse the original plan purely as a suffix and the other one purely as a prefix.

The differences of the strategies within one of the groups of repair strategies lies in the preference between agent minimization, size of preservation of the original plan and bound on the maximal length of the newly generated repairing plan component  $\pi_{\text{fix}}$ . This approach restrain bias possibly caused by unbalanced influences of the agent minimization on various types of plan repair strategies.

The approach minimizing the number of involved agents was based on the notion of a set of *supporting agents*. The iterative process from Algorithm 3 was extended with an iteration starting only with a set of agents providing at least one action, which can contribute to the repairing plan by a required proposition(s), i.e., support part of  $G \ominus \pi_{\text{suf}}$ . If such team of agents is not able to solve the plan repairing problem, the team is extended by additional agents supporting any of the current agents in the team by means of contributing to prepositions in their preconditions. If such additional agent does not exist and the team is still not containing all the agent from  $\mathcal{A}$ , a random agent is added into the team and the process continues.

### 5.1.2 Results and Discussion

The experiments were conducted in all presented planning domains and for all combinations of agent counts, i.e., two to four agents giving twelve domain and problem instances. Each of the group contained six variances of the repairing strategies giving 216 experiments in total. Each of the experiments was averaged over 5 measurements with different random seeds.

Figure 5 shows results of the first batch of experiments. The first group of repairing strategies not minimizing number of involved agents (red color) is in most measurements in both computational and communication metrics worse than the baseline replanning strategy. The suffix preserving algorithms minimizing numbers of agents (green color) is on the other hand nearly in all measurements better in both metrics than the baseline strategy with an exception in the simplest COOPERATIVE PATHFINDING problems. The group of plan repairing strategies minimizing the number of involved agents and preserving prefix part of the original plan (blue color) is on tie or better with the replanning in rather loosely coupled domains. The communication and computational overheads decrease with decreasing coupling of the domains. However in tighter

coupled domains, the strategies fall behind the replanning baseline. In LOGISTICS domain, only 33% of the strategies are better by communication overheads and only 18% by means of computational overheads. With increasing coupling the approach lose more. These results *support the second hypothesis*.

Additionally, the results revealed that the prefix preserving approaches, as not the best in all agent minimizing approaches, in most of the experiments has one of the best approaches outperforming the best suffix preserving approach. In LOGISTICS domain, the separation between the best prefix and best suffix preserving plan repairing strategy is about a half an order of magnitude in favor of the one prefix preserving approach. On the other hand, in COOPERATIVE PATHFINDING, suffix preserving approaches gain more than one order of magnitude.

## 5.2 Repairing of Actions with High Dependency Metrics

The intuition behind Hypothesis 3 can be rephrased as follows: If an action fails and it has potentially a lot of future dependencies, possibly of other agents or even in the goal, trying to fix it as soon as possible is rather better idea, than ignore it and try to repair it later. The experiments in this section were conducted to validate this concept.

### 5.2.1 Used Plan Repair

The most straightforward approach here is to compare the two plan repairing strategies re-using the whole original plan either as a prefix or as a suffix. These strategies are again parameterizations of the plan generalized repair such that there is no iteration over various  $u$  and  $v$ , but only two fixed values. The *pure prefix strategy* uses fixation  $u = |\pi| - k, v = 0$  and the *pure suffix strategy* uses fixation  $u = 0, v = |\pi| - k$ .

In order to explain the result, we have to present more details on the LOGISTICS domain. In the LOGISTICS problem with three agents used in the experiments, the agents control two trucks  $T_1$  and  $T_2$  and one airplane  $A$ . There are two cities, each with one storage depot ( $d_1$  and  $d_2$ ) and one airport ( $a_1$  and  $a_2$ ). The trucks can move  $m(\text{from}, \text{to})$  only within their cities, i.e., between one depot and one airport. The airplane can fly  $f(\text{from}, \text{to})$  among all airports in the environment, but cannot land at the depots. All vehicles can load  $l(\text{package}, \text{location})$  and unload  $u(\text{package}, \text{location})$  a package at a location. Initially, there is one package  $p$  at one of the depots and the goal is to transport it to the other depot in the other city. The trucks start at the depots and the airplane starts at one of the airports. A typical multiagent plan solving this particular instance is depicted in the matrix form in Figure 6.

### 5.2.2 Results and Discussion

To validate Hypothesis 3, we run the pure prefix and pure suffix preserving repairing strategies in all the testing domains. We have measured ratio of successful repairs of these two repairing strategies against replanning by means of computation time. In Figure 7, we summarize the results of these experiments.

In the ROVERS and SATELLITE domains the plans solving the problem do not contain any significant actions by means of number of future dependencies to the overall

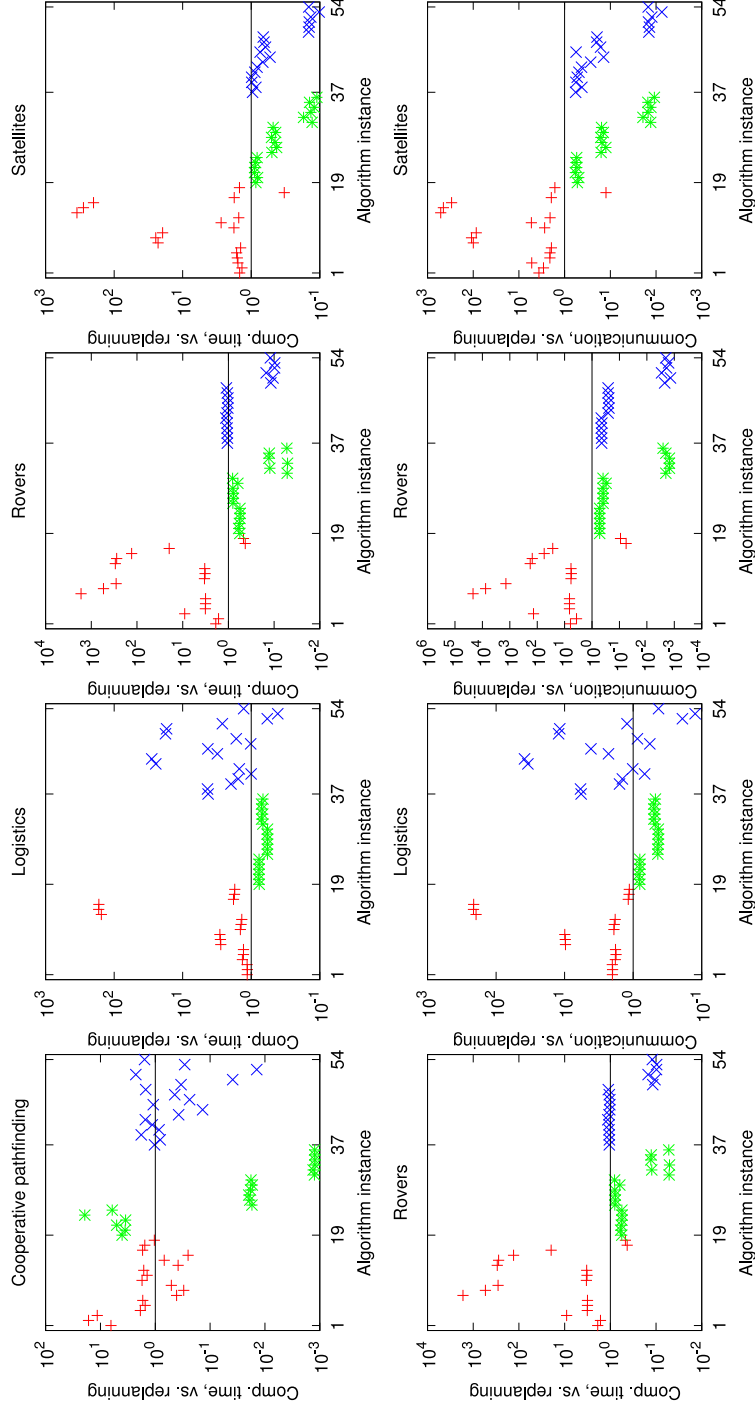


Figure 5: Comparison of various plan repairing strategies in proportion to replanning (black line at  $y = 1$ ) with failure probability  $p = 0.3$ . Each point represent a mean of several runs of one of the particular repairing strategy. The red group contains plan repairing strategies using only the full set of agents involved in the original planning problem, the green group contains strategies using minimization of the number of agents involved and preserving suffix (back-on-track) of the original plan and the blue group contains strategies also minimizing number of agents and preserving prefix (lazy repair) of the original plan.

$$\begin{array}{l}
A : \left( \begin{array}{cccccccccc}
\epsilon & \epsilon & \epsilon & \overline{l(p, a_1)} & f(a_1, a_2) & \overline{u(p, a_2)} & \epsilon & \epsilon & \epsilon \\
l(p, d_1) & m(d_1, a_1) & \overline{u(p, a_1)} & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\
m(d_2, a_2) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \overline{l(p, a_2)} & m(a_2, d_2) & \overline{u(p, d_2)}
\end{array} \right) \\
T_1 : \\
T_2 : \begin{array}{cccccccccc}
8 & 7 & 6 & 5 & 4 & 3 & 2 & 1
\end{array}
\end{array}$$

Figure 6: A multiagent plan solving the initial LOGISTICS problem used in the experiments. Empty actions are denoted as  $\epsilon$ . The overlines mark public actions. The numbers in the last row represent particular counts of steps, i.e., number of actions  $|\pi| - k$ , to the end of the plan.

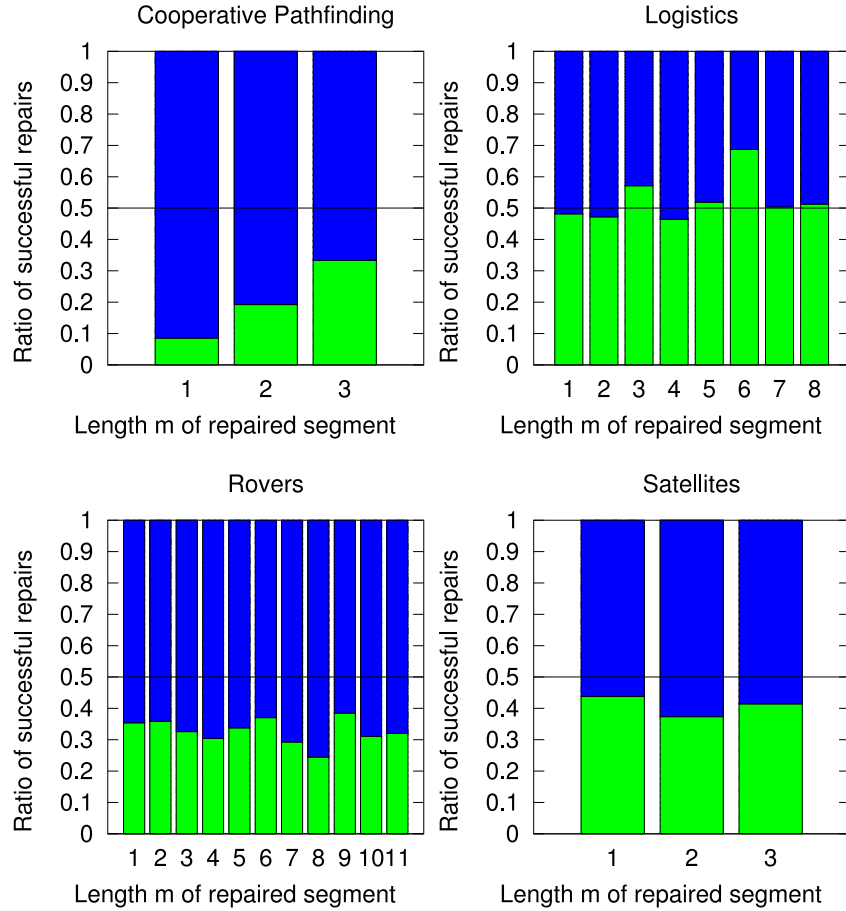


Figure 7: Comparison of success ratio against replanning between suffix preserving (green, back-on-track) and prefix preserving (blue, lazy) plan repairing with variable length  $m = |\pi| - k$  of the repaired plan segment.

count of actions in the plan. In SATELLITES, all actions are private and therefore actions of one agent depend only on other actions of the same agent. The highest dependency metrics is in such case  $\max_{a \in \bigcup_{i=1}^n A_i} \text{dep}_\pi(a) = |\pi|$  for the first action  $a$  of each satellite in the multiagent plan  $\pi$ . The individual plans of the agents are relatively short (three to four actions) and therefore the dependency metrics is never higher than four and it is zero for public projection  $\pi^{\text{pub}}$  of the plan  $\pi$ .

Multiagent plans for the ROVER problems contain several public actions at the end of the plan, representing always only one rover communicating at one time point. Although the plans solving the ROVERS problems contain public actions, there are again no long dependencies among the actions. The dependencies in the private part of the plan contain three components, each containing three to four private actions. Consequently, the private dependencies are, similarly to the SATELLITE problems, maximally four actions long. The dependencies among the public actions are even shorter, as there is the same number of public actions as agents, which means maximally three-action public dependencies for three agents. The dependency link between one public action and one dependent private component increases the maximal dependent length to maximally seven actions (four private actions of the component bound to three public actions successively dependent on each other). Using the dependency metrics  $\max_{a \in \bigcup_{i=1}^n A_i} \text{dep}_\pi(a) = 4 + n$  and  $\max_{a \in \bigcup_{i=1}^n A_i} \text{dep}_{\pi^{\text{pub}}}(a) = n$  for the public plan and for  $n$  rovers.

In such repair problem, even if one of the leading actions in a private component fail, prefix preserving (i.e., lazy) approach solves nearly the complete problem only by reusing the original plan. More precisely, it reuses the original solution for the rest of the private components and all the public actions except one of the failed agent. As the results show, the prefix-based repair is always better then the suffix-based and the ratio between these two is rather stable over different points in the plan.

The situation changes in the LOGISTICS domain. In LOGISTIC with three agents and one package, there is a chain of dependent actions. Particularly,  $u(p, d_2)$  depends on  $l(p, a_2)$ , which depends on  $u(p, a_2)$  and so on to the first action of the plan  $l(p, d_1)$ . The dependency chain has six public actions in the example plan and occupy the complete length of it. As the results show in Figure 7, there are two distinctive peaks where the suffix preserving repair outperforms the prefix preserving repair, additionally with a increasing trend. The first one is for repair plans of length  $m = 3$  and the other one is for  $m = 6$ . As presented in Figure 6, these lengths correspond to the package handover points in the plan, more precisely, to repair of failing unloads  $u(p, a_1)$  and  $u(p, a_2)$ . These public actions' dependency metrics is  $\max_{a \in \bigcup_{i=1}^n A_i} \text{dep}_{\pi^{\text{pub}}}(a) = \frac{2|\pi^{\text{pub}}|}{3}$ , which in contrast to ROVERS is dependent not only on the number of the agents, but on the length of the public plan. Ignoring a failure of unloading by the prefix preserving (i.e., lazy) approach causes the package is left in the last vehicle and the rest of the team finishes the executable remainder of the plan, which in principle means the vehicles are moving, but they are not transporting the package. On the other hand, in the same circumstances, the suffix preserving repair (i.e., back-on-track) only repeats the unload action and successfully continues with the rest of the original plan ending in a goal state.

One can argue that the complement load actions should be repaired more efficiently

using this same argumentation as well. This is very true, however this phenomenon is not captured in the results, because of a particular implementation of the MA-Plan planner. The explanation is based on the fact the used planner efficiency is more dependent on small differences in number of involved agents, than the number of planned actions. In the case of  $m = 3$  (the  $u(p, a_2)$  action), 2 agents are needed to do lazy repair, because firstly the executable remainder of the original plan is reused to the last state without the package and then the planner has to be used to generate repair plan  $\pi_{\text{fix}}$  reverting all the moves and planning to one of the goal states again. Such plan has to firstly unload the package from the airplane A and then transport it successfully by the truck  $T_2$  to the goal destination  $d_2$ . On the other hand, the pure suffix preserving approach generates only a plan repeating the unload action  $u(p, a_2)$  and afterward continues with the original plan as a suffix. This planning problem involves only one agent, in particular, the airplane A carrying out unload of the package. The same principle can be applied to  $m = 6$ , but with all three agents for pure preserving (lazy) repair, but only 2 agents for pure suffix repair (back-on-track).

In the last problem of COOPERATIVE PATHFINDING, the length of a sequence of dependent actions correspond to the length of the plan as well, as all the actions in such plan are public and inter-dependent. Nevertheless, this is quite different “order of dependency”, than in SATELLITES for example. In SATELLITES, all the actions are dependent as well, but only within one agent, whereas here, the actions are dependent across the agents. In the experimental results of the COOPERATIVE PATHFINDING a trend arises. In such dense types of inter-dependent problems, the longer are the repaired plans, the more the suffix repair algorithm gains against the prefix one.

The results of these experiments, namely of LOGISTICS and COOPERATIVE PATHFINDING, *moderately support the third hypothesis* of the paper.

### 5.3 Partitioning of the Original Plan

It is not intuitively clear what is a good strategy for reusing the original plan parts, moreover related to a particular planning domain. The experiments conducted in these sections provide several insights into this issue and focus especially on answering the Hypothesis 4.

#### 5.3.1 Used Plan Repair

A battery of plan repairing strategies was prepared to validate Hypothesis 4. We parameterize how much generalized repair reuse the original plan. Such parameterization (based on the  $U$  and  $V$  indice vectors) lead to a two-dimensional discrete space of different plan repairing strategies, as depicted in Figure 8, representing a structure of the repaired plan.

Each of the nine diagrams in the figure describes a variation on a resulting plan repaired by one particular parameterization of the algorithm in the context of execution of the original plan. The execution starts with a world in the initial state  $I$  and it is anticipated to continue with help of the original plan to the last state  $S_m$ , which is one of the goal states, i.e.,  $S_m \supseteq G$ . However during execution of an action following a state  $S_k$ , execution failed and the state of the world ends up not in the state  $S_{k+1}$ , but



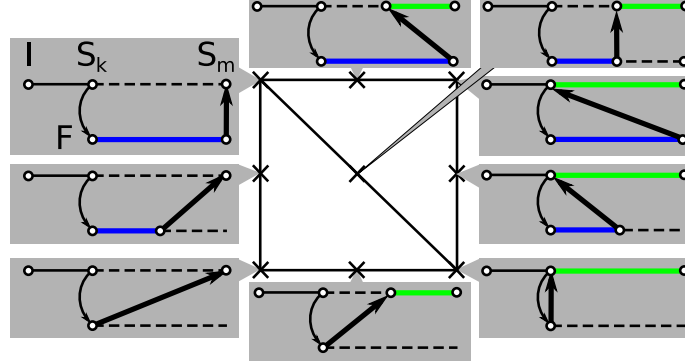


Figure 8: Scheme of a two-dimensional space representing plan repairing strategies preserving different parts of the original plan and reusing it in different ways. The blue segments represent prefix re-usage and the green ones the suffix re-usage. The notable states are: initial state  $I$ , last achieved state  $S_k$  induced by the original plan, exceptional state  $F$  after a failure and the last anticipated state  $S_m \supseteq G$ , provided that the original plan would be executed without a failure.

in a state  $F$ , out of the anticipated sequence of states and actions. To fulfill the goal, the agents use one of the plan repairing strategies, which under the condition of perfect execution, would transform the world from  $F$  to a  $S_m \supseteq G$ .

In Figure 8, there are two dimensions depicted. One of the dimensions represent the number of actions which has to be reused from beginning of the original plan as a prefix corresponding to fixation of the iteration parameter  $U = (|\pi| - k)$ . The other dimension represents number of actions re-used as suffix of the final repairing plan, i.e., fixed iteration parameter  $V = (|\pi| - k)$ . In the presented scheme,  $\pi_{\text{pre}}$  from the Algorithm 3 is denoted as a blue line,  $\pi_{\text{suf}}$  as a green line and  $\pi_{\text{fix}}$  as a black thick arrow. Since both the dimensions reuse the same original plan, the space is always a square with a side of the length  $|\pi| - k$ .

There are four extremes in the repair strategy space. The strategy at position  $(0, 0)$  effectively degenerates from  $\pi_{\text{pre}} \cdot \pi_{\text{fix}} \cdot \pi_{\text{suf}}$  to  $\pi_{\text{fix}}$ . Such process correspond to replanning from the scratch. The strategies at positions  $(|\pi| - k, 0)$  and  $(0, |\pi| - k)$  represent pure repairs  $\pi_{\text{pre}} \cdot \pi_{\text{fix}}$  and  $\pi_{\text{fix}} \cdot \pi_{\text{suf}}$  respectively. The last extreme at  $(|\pi| - k, |\pi| - k)$  represent an strategy, which firstly uses the original plan ignoring inapplicable actions, then using a newly generated plan  $\pi_{\text{fix}}$  returns to the anticipated state after execution of the failed action  $S_k$  and than it reuses the original plan again to get to the goal state, i.e., the algorithm generates a full overlap of the prefix and suffix plans.

Beside the extremes, also the  $(0, m), (1, m-1), \dots, (m-1, 1), (m, 0)$  diagonal for  $m = |\pi| - k$  in the space is important from perspective of the ongoing discussion. All the strategies lying on this diagonal can re-use all the actions of the original plan exactly once and in the original order. Meaning, the original plan is neither overused nor underused. Formally, we define:

**Definition 12.** (*m-normal generalized plan repair*) Let  $\Sigma = \langle \Pi, \pi, F, k \rangle$  be a multia-

gent plan repairing problem, then an algorithm  $R$  is a *m-normal generalized plan repair*, iff  $R$  solves the problem  $\Sigma$  by a multiagent plan  $\pi'$  with decomposition  $\pi_{\text{pre}} \cdot \pi_{\text{fix}} \cdot \pi_{\text{suf}}$  and at the same time  $(|\pi_{\text{pre}}|, |\pi_{\text{suf}}|) \in \{(0, m), (1, m-1), \dots, (m-1, 1), (m, 0)\}$ , where  $m = |\pi| - k$ .

### 5.3.2 Results and Discussion

To validate the third and last hypothesis, we used a randomized sampling of the strategy space and searched for more successful algorithms lying on the *m*-normal diagonal. The results are present in Figure 9.

The sampling experimental process measured for each encountered repairing problem the computation time of the replanning strategy. After this base-line measurement, a tested repairing strategy was run with a bound on the computation time based on the replanning run-time. If the strategy performed better, a cell in the result map was incremented by one. In effect, this process rendered the presented normalized results. During the experimental execution and plan repairing, we used different lengths of the original plan, i.e., the repair was done for various  $|\pi| - k$ . Therefore, the resulting maps depict a continuous space, as the results with higher and lower  $|\pi| - k$  values were merged into the most representative *m* value corresponding to the initial multiagent plan generated.

As the maps show, the hypothesis clearly holds for coupled domains with longer plans (LOGISTICS, and ROVERS). In the coupled domain of COOPERATIVE PATHFINDING, the diagonal is also present, but because of shorter repaired plans, it degenerated considerably. In the experiment with SATELLITES, the diagonal is not present.

These results *support* Hypothesis 2 with an auxiliary observation, that the effect is decreasing as the coupling of the domain decreases.

## 6 Related Work

There are several approaches capable to drive multiagent team activities in an environment with uncertain dynamics.

Firstly, there is a body of literature dealing with and extending models of decentralized partially observable Markov Decision Processes (Dec-POMDPs) [1]. A Dec-POMDP model leads to computation of a *policy* for the agents in the environment ensuring that by following it, the team reaches (joint) rewards. The model assumes only partial observability of the environment and it is capable of capturing various eventualities which can occur in the environment. The eventualities, however, have to be probabilistically known *a priori*, such that a model of action outcomes can be constructed before planning. Dec-POMDP solvers do not scale well to larger problems, especially when the model of run-time action failures is *a priori* unknown. The plan repairing algorithms proposed in this article do not require an explicit failure model as an input. The price the plan repairing algorithms pay is their inefficiency in problems with failures taking the system far from the presumed evolution based on the original plan and rate of such failures as shown in [10]. A simplified single-agent models described by Markov Decision Processes (MDP) were tackled by scalable techniques of online

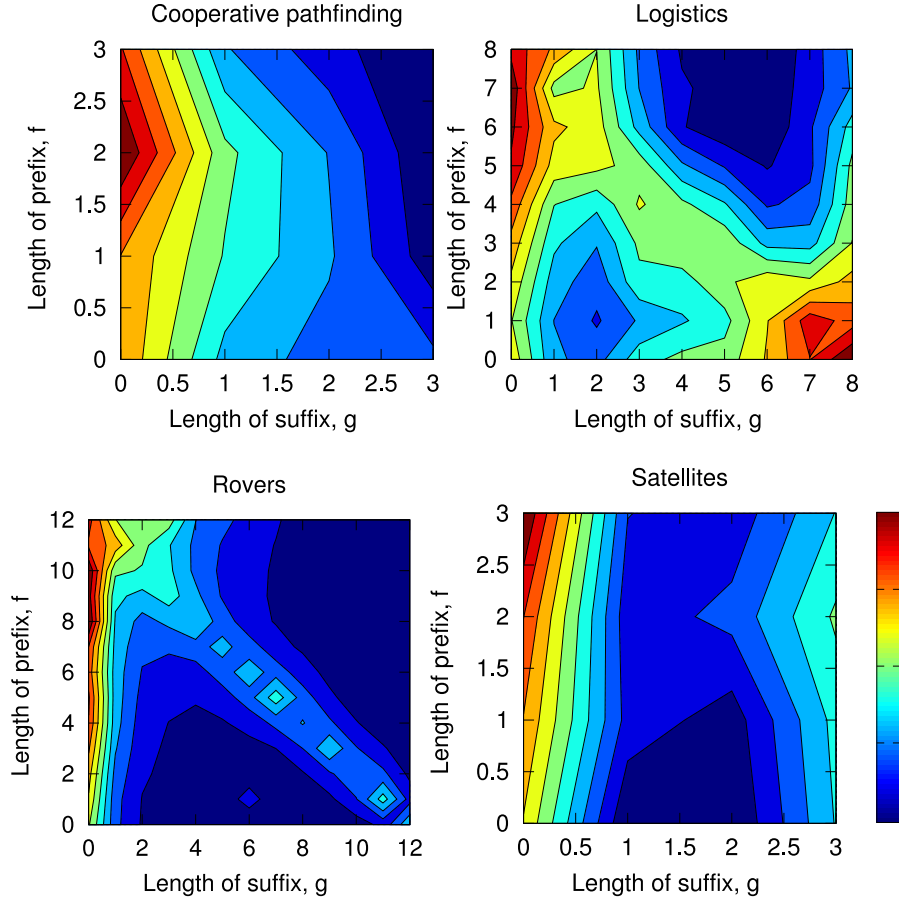


Figure 9: The maps present prefix ( $u$  on  $y$ -axis) vs. suffix ( $v$  on  $x$ -axis) preserving repairing algorithms by a success rate against replanning in the repair time for all domains with three agents and  $p = 0.3$ . Red color represents repair strategies faster than replanning. The top-left to bottom-right diagonal represent algorithms neither overusing or underusing the original plan.

policy replanning by problem determinization e.g., in FF-Replan [16]. Determinization approaches are related to the online replanning scheme. Since we do not assume the probabilistic model of the failures for the multiagent plan repair algorithms, we cannot prepare the (partial) policies using the determinization of the model. Our approach focuses on efficiency of the replanning process per se, when it is needed because of a failure. In other words, plan repair assumes an dynamic and optimistic determinization of the problem based on the original plan.

Secondly, single-agent Contingency [11], Fault Tolerant [7] and Conformant [14] planning techniques facilitate classical-style planning for domains with non-probabilistic uncertainty in either action outcomes or state the system happens to be in. However, again, in order to plan for actions in such domains, the possible contingencies and action models in the environment must be known before the planning phase. In the multiagent plan repair, it is assumed any possible outcome of an action in general. In Contingency, Fault Tolerant, or Conformant planning such assumption would lead to actions possibly taking the environment to any state. Thus a complete graph representing all transitions would render the techniques unusable.

Lastly, the idea of macro actions used for single-agent plan repair [15] build upon the positive results of planning with prescribed sequences of primitive actions. The technique stemmed from the area of integrating planning and machine learning [4] and was adapted to describe parts of the repaired plan by fixed macro actions. In respect to the recency of the techniques, it is not surprising that it was not extended for multiagent planning yet.

## 7 Conclusion

Based on the theoretical and experimental results, we can come up with a summary of heuristic approaches in form of simply usable advices decreasing computation and/or communication overheads during repairing of multiagent plans. These advices can be used for various plan repairing approaches targeting systems with planning agents reusing the original plan in form of combination of prefix and suffix as we proposed in the generalized repair. The results were verified for plan repairing techniques utilizing preservation of the original plan and using an CSP-based multiagent planner to fill prospective discontinuities in the repairing plan. The advices are:

1. *Use plan reuse based on generalized repair only with plans of polynomially bounded length.*
2. *Prefer smaller numbers of involved agents in the plan repairing process.*
3. *Prefer prefix preserving repairing techniques when repairing failures with long dependencies among different agents.*
4. *Prefer  $m$ -normal generalized plan repairing algorithms.*

This work opens several interesting questions left for the future work. Most notably, how would another implementation of the underlying multiagent planner affect the results and would it be possible to integrate principles from single-agent search effort estimation approaches, e.g., as in [6] to provide more precise hints how to repair during the execution and repairing process.

## Acknowledgments

This research was partly supported by the Czech Science Foundation (grant no. 13-22125S) and partly by USAF EOARD (grant no. FA8655-12-1-2096).

## Bios

*Antonín Komenda* is a research fellow at Agent Technology Center of Czech Technical University (CTU) in Prague since 2007. He holds PhD from Artificial Intelligence and Biocybernetics from CTU with thesis on Domain-independent Multiagent Plan Repair. In his research, he focuses on theory, methods and algorithms for automated multiagent planning, distributed problem solving, coordination and robust multi-agent planning. In 2010, Antonin undertook a 5-week internship at Drexel University in Philadelphia in the group of prof. Regli and in 2013–2014, he was a post-doctoral fellow at Technion – Israel Institute of Technology in Haifa in the group of prof. Domshlak.

*Peter Novák* is a post-doctoral research fellow in the Algorithmics group at the Delft University of Technology in the Netherlands, in the group of Cees Witteveen since March 2012. Before moving to Delft, he spent more than 2 years as a post-doctoral research fellow in the Agent Technology Center (ATG) of the Czech Technical University in Prague, in the group of Michal Pěchouček. Peter holds the doctoral degree in computer science from Clausthal University of Technology, where he worked in the Computational Intelligence Group under supervision of Jürgen Dix. Peter's long-term research interests are rooted in investigation of interactions of agents with dynamic environments; more specifically, cognitive robotics, (multi-)agent oriented programming of cognitive, or knowledge-intensive agents and multi-agent planning and plan-repair with a strong interest in practical applications of agent-oriented technologies.

*Michal Pěchouček* is a full professor at the Czech Technical University in Prague, Head of Agent Technology Center, Deputy head for research at Department of Computer Science, Director of Open Informatics Programme. In 2011 visiting professor at University of Southern California, TEAMCORE lab (Fulbright fellow), in 2006 visiting scientist at University of Edinburgh (Royal Society of Edinburgh), in 2003 visiting professor at State University of New York at Binghamton, in 2000 postdoc researcher at University of Calgary. Chair of the board of directors for EURAMAS, member of the board of directors of IFAAMAS. Visiting/honorary member of the Artificial Intelligence Application Institute, University of Edinburgh, member of the advisory board of the Center for Advanced Information Technologies, SUNY Binghamton. Member of the scientific council at Masaryk University, School of Informatics, Member of Czech Science Foundation Panel on Computer Science.

## References

- [1] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, November 2002. <http://dx.doi.org/10.1287/moor.27.4.819.297>.
- [2] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS*, pages 28–35, 2008. <http://www.aaai.org/Library/ICAPS/2008/icaps08-004.php>.
- [3] Steve Chien, Minh Binh Do, Alan Fern, and Wheeler Ruml, editors. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*. AAAI, 2014. <http://www.aaai.org/Library/ICAPS/icaps14contents.php>.
- [4] Lukás Chrpá, Mauro Vallati, and Hugh Osborne. Learnability of specific structural patterns of planning problems. In *ICTAI*, pages 18–23. IEEE, 2013. <http://dx.doi.org/10.1109/ICTAI.2013.14>.
- [5] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. <http://www.elsevier.com/wps/find/bookdescription.agents/678024/description>.
- [6] Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger H. Hoos, and Kevin Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In Chien et al. [3]. <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7939>.
- [7] R. M. Jensen, M. M. Veloso, and R. E. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *ICAPS*, pages 335–344, 2004. <http://www.aaai.org/Library/ICAPS/2004/icaps04-040.php>.
- [8] Antonín Komenda, Peter Novák, and Michal Pěchouček. Decentralized multi-agent plan repair in dynamic environments (Extended Abstract). In *Proceedings of AAMAS*, pages 1239–1240, 2012. <http://dl.acm.org/citation.cfm?id=2343941&dl=ACM&coll=DL&CFID=612050604&CFTOKEN=59981303>.
- [9] Antonín Komenda, Peter Novák, and Michal Pěchouček. How to repair multi-agent plans: Experimental approach. In *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 2013. <http://icaps13.icaps-conference.org/wp-content/uploads/2013/05/dmap13-proceedings.pdf>.

- [10] Antonín Komenda, Peter Novák, and Michal Pěchouček. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*, 2013. <http://www.sciencedirect.com/science/article/pii/S1084804512002585>.
- [11] Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In Carla E. Brodley and Peter Stone, editors, *AAAI*, pages 2322–2329. AAAI Press, 2014. <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8656>.
- [12] B. Nebel and J. Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2):427–454, July 1995. <http://www.sciencedirect.com/science/article/pii/000437029400082C>.
- [13] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS*, pages 1323–1330, 2010. <http://dl.acm.org/citation.cfm?id=1838379>.
- [14] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Int. Res.*, 35(1):623–675, August 2009. <http://dl.acm.org/citation.cfm?id=1641503.1641518>.
- [15] Enrico Scala. Plan repair for resource constrained tasks via numeric macro actions. In Chien et al. [3]. <http://www.aaai.org/Library/ICAPS/icaps14contents.php>.
- [16] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, page 352. AAAI, 2007. <http://www.aaai.org/Library/ICAPS/2007/icaps07-045.php>.

# The MADLA Planner: Multiagent Planning by Combination of Distributed and Local Heuristic Search

Michal Štolba<sup>a</sup> and Antonín Komenda<sup>b</sup>

<sup>a</sup> *Department of Computer Science, Faculty of Electrical Engineering,  
Czech Technical University in Prague  
Karlovo náměstí 13, Praha 2, 121 35, Czech Republic  
michal.stolba@agents.fel.cvut.cz, +420 22435 7693, **corresponding author***

<sup>b</sup> *Department of Computer Science, Faculty of Electrical Engineering,  
Czech Technical University in Prague  
Karlovo náměstí 13, Praha 2, 121 35, Czech Republic  
antonin.komenda@agents.fel.cvut.cz*

---

## Abstract

Multiagent planning for the MA-STRIPS model requires a process of distributed plan generation for a cooperative team of agents. Heuristic multiagent state-space search is an obvious candidate providing scheme both for agents' local search and inter-agent protocol for distribution of the search. However, distributed heuristic estimation, application of multi-heuristic search and efficient utilization of the decentralized computation power are still open challenges.

The Multiagent Distributed and Local Asynchronous (MADLA) Planner runs a distributed variant of a state-space forward-chaining multi-heuristic search with two versions of a well known FastForward relaxation heuristic, one estimating the particular agent's local subproblem and another estimating the global heuristic values. We propose a general asynchronous scheme for such multi-heuristic multiagent searches and provide proofs of soundness and completeness. The asynchronism allows efficient utilization of agents' computation power while waiting for responses from other agents. Also, we provide a novel distribution scheme for the FastForward heuristic, inspired by the Set-Additive variation of the FastForward heuristic and with lazy computation of partial relaxed plans.

We experimentally compare the proposed multi-heuristic scheme and the two used heuristics per se. The results show the proposed solution outperforms search with either one of the heuristics used separately, positively combining benefits of both heuristics. In the detailed experimental analysis, we show limits of the planner and of the used heuristics based on particular properties of the benchmark domains. In a comprehensive set of multiagent planning domains and problems, we show that the MADLA Planner outperforms all state-of-the-art MA-STRIPS multiagent planners.

*Keywords:* multiagent planning, automated planning, multiagent systems, state-space search, multi-heuristic search, MA-STRIPS

---



## 1. Introduction

The ability of machines to deliberate about sequences of actions—plans—is one of the classical areas of artificial intelligence. In the case of multiple machines, or agents, interacting both in the plan synthesis and execution phase, we talk about multiagent planning (MAP). In contrast to distributed plan synthesis, in the multiagent case the agents are restricting their computational processes to communicate only information identified as public and not reveal all the information they are working with. Such restriction can be beneficial not only in domains requiring it by definition, but also to increase efficiency by focusing only on subproblems relevant for particular agents.

The recently most prevalent model for MAP goes back to the roots of research in planning itself—to STRIPS. The STRIPS model [1] formalized planning by a propositional description of the world and the actions, together forming a transition system representing the planning problem. An extension of STRIPS towards MAP denoted as MA-STRIPS [2] generalizes the model by allowing more than one finite set of actions, each set characterizing capabilities of one agent. Since not all agents must necessarily be capable of influencing the whole environment, some parts of the information about it can be private to a subset of agents, alongside the common information treated as public to all agents.

Because of its computational complexity [2], MAP planners rely, similarly to classical planning, usually on automatically derived heuristic functions and algorithms estimating cost to a goal state. The heuristic state-space search in MAP was proposed as a form of well-known A\* search algorithm, denoted as Multiagent Distributed A\* (MAD-A\*) in [3, 4] and as Multiagent Best-First Search (MA-BFS) in [5, 4].

One of the most prominent classical planning heuristics still in use is Fast-Forward (FF) [6], first introduced in the FF planning system. In MA-STRIPS literature, classical heuristics were used in the MAD-A\* planner, but restricted only to *projected* problems, that is each particular agent was using the heuristics only on the part of the MAP problem it has access to. Such projection can substantially underestimate the cost as it does not consider any other agents' private actions. A seeming remedy is to *distribute* the process of heuristic estimation among all agents such that each agent preserves its privacy by not communicating anything else than its partial heuristic estimations. A distributed estimator for the FF heuristic, proven to return the same estimations as centralized FF, is MA-FF [7] (based on building a distributed form of Relaxed Planning Graphs [8]). MA-FF is, however, practically inefficient, more precisely, the results indicates that there is a trade-off between the quality of the heuristic estimation and the efficiency of distributed heuristic computation, following similar results in classical planning with an additional communication overhead. Two distributed variants of MA-FF focused on practical improvement, namely lazily computed lazyFF [7, 5], and rdFF using recursive distributed computation [5]. Although both heuristics improves efficiency of the underlying search in contract to MA-FF, the projected variant of FF still performs better in non-negligible

number of planning problems, because of its computational ease and no communication requirements [5]. A natural solution is to combine the projected and distributed variants of FF.

The principle of combining more heuristic estimators is well known in classical planning as the multi-heuristic search [9, 10], yet, it was never analyzed in the context of MAP. Specifically, the particular way, how to combine projected and distributed heuristic estimators in multiagent distributed state-space search resulting in an efficient planning approach, is an open problem, as the methods used in classical planning are clearly not suitable for this class of heuristics. In this paper, we address this issue both in theory using a general model and in practice using a particular implementation of a planner. The same general principle can be easily applied also in areas of distributed AI other than multiagent planning.

## 2. Multiagent Planning

As the Multiagent Distributed and Local Asynchronous (MADLA) planner is based on the MA-STRIPS planning model, we reuse most of the formal definitions from [2] and amend them mostly with formalization of state and action projections required in later definitions of the proposed search scheme and heuristics.

We assume a set of  $n$  *cooperative* agents with common goals which search for a multiagent plan solving a planning problem in a *coordinated* fashion. The search is decoupled from the prospective execution of the plan, similarly as is classical (off-line) planning. A multiagent planning problem is formally defined as:

**Definition 1.** Let a quadruple  $\Pi = \langle P, \{A_i\}_{i=1}^n, I, G \rangle$  be a MA-STRIPS planning problem for  $n$  agents  $\mathcal{A} = \{\alpha_i\}_{i=1}^n$ , where:

- $P$  is a finite set of propositions describing facts about the world the agents act in, a state of the world will be denoted as  $s \subseteq P$ ,
- $A_i$  is a finite set of actions an agent  $\alpha_i$  can perform
  - each action  $a \in A = \bigcup_{i=1}^n A_i$  has the standard STRIPS syntax and semantics  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$ , where  $\text{pre}(a) \subseteq P$ ,  $\text{add}(a) \subseteq P$ ,  $\text{del}(a) \subseteq P$  represent preconditions, add effects, and delete effects respectively,
  - a transition function  $\text{apply} : 2^P \times A \rightarrow 2^P$  is defined as  $\text{apply}(a, s) \mapsto s'$  provided that  $\text{pre}(a) \subseteq s$  s.t.  $s' = s \cup \text{add}(a) \setminus \text{del}(a)$ , where action  $a$  is applied in state  $s$  with a new resulting state  $s'$ ,
  - the sets of actions are pairwise disjoint, that is  $\forall i \neq j : A_i \cap A_j = \emptyset$ ,
- $I \subseteq P$  is the initial state of the world, and

- $G \subseteq P$  is the goal condition defining the goal (final) states of the problem; a state  $s$  is a goal state iff  $G \subseteq s$ .

A set of all actions of all agents will be denoted as  $A = \bigcup_{i=1}^n A_i$ . Subsequently, a *global planning* problem of a multiagent planning problem  $\Pi$  will be defined as  $\Pi^G = \langle P, A, I, G \rangle$ .

In order to define a set of fact restricted to one agent, we firstly define a set  $\text{facts}(a) = \text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)$  which denote facts required and/or affected by an action  $a$ . Then all facts of agent  $\alpha_i$  are defined as  $P_i = \bigcup_{a \in A_i} \text{facts}(a)$ .

MA-STRIPS provides a scheme for separation of private (internal) information of particular agents and public (common) information which is shared among all agents. Private facts of an agent  $\alpha_i$  called  $\alpha_i$ -internal are its facts which are not facts of any other agent, formally  $P_i^{\text{int}} = P_i \setminus \bigcup_{\alpha_j \in \mathcal{A} \setminus \alpha_i} P_j$ . Public facts of an agent  $\alpha_i$  are the complement  $P_i^{\text{pub}} = P_i \setminus P_i^{\text{int}}$ . In a similar manner, we define separation of private and public actions of the agents. Private actions of agent  $\alpha_i$  are such actions that does not affect and are not affected by other agents via the public facts, formally

$$A_i^{\text{int}} = \{a | a \in A_i, \text{facts}(a) \subseteq P_i^{\text{int}}\}.$$

Conversely, public actions are  $A_i^{\text{pub}} = A_i \setminus A_i^{\text{int}}$ .

States and actions restricted to a specific set of facts are projections on that specific set. If a projection is on a set of facts of an agent  $\alpha_i$  and all public facts, formally  $P_i^{\text{proj}} = P_i \cup \bigcup_{i \neq j} P_j^{\text{pub}}$ , we will be using the term  $\alpha_i$ -projection of a state, formally defined as  $s^{\alpha_i} = s \cap P_i^{\text{proj}}$ . Similarly, an  $\alpha_i$ -projection of an action  $a \in A_i$  is defined as

$$a^{\alpha_i} = \langle \text{pre}(a) \cap P_i^{\text{proj}}, \text{add}(a) \cap P_i^{\text{proj}}, \text{del}(a) \cap P_i^{\text{proj}} \rangle.$$

A solution of multiagent planning problem is a multiagent plan.

**Definition 2.** Let a sequence of actions  $\pi = (a_1, \dots, a_k)$  be a *multiagent plan* solving a multiagent planning problem  $\Pi = \langle P, \{A_i\}_{i=1}^n, s_0, G \rangle$  iff the sequence is *sound* that is  $\text{pre}(a_i) \subseteq s_{i-1}$ , where  $s_i = \text{apply}(s_{i-1}, a_i)$  for  $i \in 1, \dots, k$  and  $G \subseteq s_k$ .

In the rest of the article, we will use the term *plan* for multiagent plans and in ambiguous cases we will distinct multiagent plans and single agent plans. Additionally, we will use the term *dead-end* state  $s$  to indicate that no plan exists from  $s$  to any goal state.

A computational process generating solutions for planning problems is a planner.

**Definition 3.** A (distributed) algorithm is a *multiagent planner* iff it accepts a multiagent planning problem  $\Pi = \langle P, \{A_i\}_{i=1}^n, I, G \rangle$  as an input and produces a multiagent plan  $\pi$  solving  $\Pi$  as an output. Such planner is

**sound** iff any produced multiagent plan  $\pi$  of the multiagent planner is a sound plan for  $\Pi$ ,

**complete** iff the multiagent planner produces a plan for any multiagent problem  $\Pi$  s.t.  $\exists \pi$  which is solution of  $\Pi$ ,

Similarly as in definition of multiagent plan, we will use the term planner for multiagent planners in unambiguous cases. Additionally, we assume that  $c(a) \mapsto 1$ , i.e., the optimality criterion is the number of the actions in the plan, or *length*  $|\pi| = k$  of a plan  $\pi = (a_1, \dots, a_k)$ , but all the presented techniques are easily modified to the general case. A complement to the definition of a cost function is a definition of a *heuristic function*  $h^\Pi(s)$  for a problem  $\Pi$  as  $h^\Pi : 2^P \rightarrow \mathbb{R}^+$  estimating the cost of a plan from a state  $s$  to a goal state. An evaluation  $h^\Pi(s) \mapsto \infty$  will denote states  $s$  which the heuristic  $h^\Pi$  estimates as dead ends. A *heuristic estimator* of a heuristic function  $h$  is an algorithm evaluating the heuristic  $h$ .

The definition of a multiagent planner wraps up the formalization for the context of multiagent planning in the rest of the article.

### 3. The MADLA Planner

The MADLA planner proposed in this article is a sound and complete MA-STRIPS multiagent planner (Definition 3). The planner distributively searches through a state-space induced by a multiagent planning problem  $\Pi$  (Definition 1) and if a solution exists, it returns a multiagent plan  $\pi$  solving  $\Pi$  (Definition 2). The search is driven by a combination of two heuristics  $h_L$  and  $h_D$ , the first one  $h_L$  works on single agent's view (projection) of the problem  $\Pi$  and the other  $h_D$  works distributively over the global problem  $\Pi^G$ .

Only a specific pair of heuristics comply with the proposed search<sup>1</sup>.

**Definition 4.** A heuristic estimator of a heuristic function  $h$  for a planning problem  $\Pi^G$  run by agent  $\alpha_i$  is *non-blocking* iff the computation of  $h$  does not block the computation process of agent  $\alpha_i$  for the whole duration of the computation of  $h$ .

For example a global heuristic estimator can require computation of parts of the heuristic estimate by other agents. If such heuristic algorithm is non-blocking, it does not wait for responses from other agents and allows the agent to run asynchronously while waiting for the responses, that is, the agent can use the time to perform some other computations.

A definition of dominance is the same as in the classical planning:

---

<sup>1</sup>The answer to question “why” will be explained in Section 3.2 describing the MADLA search in detail.

**Definition 5.** A heuristic function  $h_1$  *dominates* a heuristic function  $h_2$  for a planning problem  $\Pi$  iff for all states  $s \in 2^P$  hold that  $h_1(s) \geq h_2(s)$ .

Finally, we need a relation between two heuristics describing their relative computational hardness:

**Definition 6.** A heuristic estimator of a heuristic function  $h_1$  is computationally *harder* than heuristic estimator of a heuristic function  $h_2$  for a planning problem  $\Pi$  iff for all states  $s \in 2^P$  holds that computation of  $h_1(s)$  takes the same or longer time than computation of  $h_2(s)$ .

With the help of the three definitions, we can define properties of a pair of heuristics which is required for the search in the MADLA planner:

**Definition 7.** For a multiagent planning problem  $\Pi$ , let  $h_L$  be a heuristic function for  $\Pi$  and let  $h_D$  be a heuristic functions for the global problem  $\Pi^G$ . The heuristics  $h_L$  and  $h_D$  are MADLA heuristic pair iff  $h_D$  uses a non-blocking estimator by Definition 4,  $h_D$  dominates  $h_L$  by Definition 5, and  $h_D$  estimator is computationally harder then  $h_L$  estimator by Definition 6.

The intuition behind the properties is that the search combines a computationally fast heuristics  $h_L$  working not necessarily with all the information in the problem therefore likely with worse estimation and a computationally slower heuristics  $h_D$  working with global information such that the agent using the estimator can get additional information from other agents (e.g. in the form of increasing cost of some actions). Therefore the global heuristic estimate is higher than the local estimate. Such combination can work smoothly only if  $h_D$  does not block the agent using it, therefore the last property required allows the agents to continue the search using  $h_L$  whilst waiting for responses from other agents computing parts of  $h_D$ .

In next sections, we describe the heuristics and the MADLA search in details.

### 3.1. Heuristics

The key principle behind the MADLA planner is a favorable combination of both local and distributed heuristics. A heuristic pair complying with the used search in the MADLA planner combines a light single-agent and a heavy distributed multiagent heuristics which in this article are both forms of the FastForward (FF) heuristics [6]. Here, we describe both variants and show that they satisfy the properties required by Definition 7.

#### 3.1.1. Local Heuristic

In this section, we will introduce the local heuristic used in our planning system. By “local” we mean a heuristic, which computes the estimates using only the respective agent’s view of the problem as proposed in [3] (also referred to as projected heuristic, as the heuristic estimate is for the projected problem).

In MA-STRIPS, for agent  $\alpha_i$ , this means using only its  $\alpha_i$ -internal and public facts, its actions and  $\alpha_i$ -projections of actions of other agents  $\bigcup_{\alpha_j \in \mathcal{A}} \{a^{\alpha_i} | a \in A_j\}$ . A resulting local planning problem is the multiagent planning problem  $\Pi$  projected to agent  $\alpha_i$ .

**Definition 8.** Let  $h^{\Pi^{\alpha_i}} : 2^{P_i} \rightarrow \mathbb{R}^+$  be a *local heuristic function* for a projection  $\Pi^{\alpha_i} = \langle P_i, \bigcup_{\alpha_j \in \mathcal{A}} \{a^{\alpha_j} | a \in A_j\}, I \cap P_i^{\text{proj}}, G \cap P_i^{\text{proj}} \rangle$  of a multiagent planning problem  $\Pi$  for agent  $\alpha_i$ .

*Local FF Heuristic.* One of the most successful and most studied heuristics for satisficing planning is the FF heuristic [6]. We use FF in the MADLA planner as the local heuristic. The FF heuristic belongs to the delete relaxation heuristic family.

The idea behind delete relaxation heuristics is to simplify the problem by ignoring negative effects of actions. In the STRIPS formalism, this means that an action  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$  is transformed to a relaxed form  $a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$ . A set of relaxed actions denoted  $A^+ = \{a^+ | a \in A\}$  is used in definition of a relaxed planning problem  $\Pi^+ = \langle P, A^+, I, G \rangle$  respective to the original planning problem  $\Pi = \langle P, A, I, G \rangle$ . Finding an optimal plan  $\pi^+$  for  $\Pi^+$  is NP-Complete [11], therefore it is still impractical heuristic estimation. In order to lower the complexity even more, approximations of  $\pi^+$  are used in classical planning. The most commonly used approximation is a sub-optimal relaxed plan (RP). Finding a sub-optimal RP can have as low as polynomial complexity and therefore can be fast enough in practice. The length of RP is used as the heuristic estimate.

The main idea behind the FF heuristic is to perform analysis of which facts are successively reachable by applied relaxed actions (reachability analysis) and from this analysis to determine the relaxed plan in a backward fashion. The principle uses a notion of a *supporter action*  $a$  of fact  $p$  which is an action  $a$  s.t.  $p \in \text{add}(a)$ . Let  $\Pi^+ = \langle P, A^+, I, G \rangle$  be a relaxed planning problem, then the principle is following:

1. Initialize a set of unsupported facts  $U$  to contain all goal facts:  $U \leftarrow G$ .
2. Move an unsupported fact  $p$  from  $U$  to a set of supported facts  $S$  and determine its supporter  $a$ .
3. Mark all preconditions of  $a$  as unsupported if not supported already:  $U \leftarrow U \cup (\text{pre}(a) \setminus S)$ .
4. Loop 1-3 until all facts in  $U$  are either supported or in the initial state: until  $U \setminus S = \emptyset \vee U \subseteq I$ .

There are many methods how to implement this high-level scheme (which differ mainly in the way how the supporters are chosen) and many methods how to perform the reachability analysis. In our work, we have used one of the most prevalent, based on an Exploration Queue algorithm as implemented for example in the Fast-Downward planning system [9]. The main principle of Exploration Queue is that reachable facts are iteratively added to the queue and as the head of the queue is removed, if some action becomes applicable (the removed fact was its last unsatisfied precondition and not yet removed), the action is applied and its effects added to the queue.

The local FF heuristic is used in MADLA on the projection of the planning problem  $\Pi$  in form of  $h^{\Pi^{\alpha_i}}$  by Definition 8. The relaxation is therefore done

over actions  $A_i$  of an agent  $\alpha_i$  such that the problem is local relaxation formally

$$\Pi^{\alpha_i+} = \left\langle P_i, \bigcup_{\alpha_j \in \mathcal{A}} \{a^{\alpha_i+} | a \in A_j\}, I \cap P_i^{\text{proj}}, G \cap P_i^{\text{proj}} \right\rangle. \quad (1)$$

### 3.1.2. Distributed Heuristic

In addition to a local heuristic, the MADLA planner uses a distributed multiagent heuristic (also referred to as global heuristic, as the heuristic estimate is for the global problem). In contrast to the computation of a local heuristic by a single agent, a coordinated computation by multiple agents is necessary for evaluation of distributed heuristic estimates. Computation of such heuristic incurs a communication overhead, which is in many cases outweighed by better estimations w.r.t the local form of the heuristic.

In MA-STRIPS, a distributed heuristic is defined over a multiagent planning problem:

**Definition 9.** Let  $h^\Pi : 2^P \rightarrow \mathbb{R}^+$  be a *distributed heuristic function* for a multiagent planning problem  $\Pi$ .

Even if the amount of communicated bytes is not of a concern, the process of communication is multiple times slower than local computation and thus intensive communication can significantly slow down the speed of the planner. This problem was tackled in the literature by computing the distributed parts upfront and *caching* the results [12], or by *lazy* (on-request) evaluation [7, 5]. In both our recent heuristics and a new heuristic proposed in this article we use the latter approach.

*Distributed FF Heuristic.* The novel heuristic in the next section is based on our distributed lazily evaluated form of the FF heuristic, which was presented in [7] using explicit Relaxed Planning Graphs for the reachability analysis and updated to use the more efficient Exploration Queue algorithm in [5]. This approach, termed Lazy FF (lazyFF), distributes the process of finding and extracting Relaxed Plan from a state  $s$  based on the following principle:

1. The agent  $\alpha_i$  initiating the estimation locally computes a projected relaxed plan  $\pi^{\alpha_i+}$  which is a solution of an  $\alpha_i$ -projection of the relaxed problem  $\Pi^{\alpha_i+}$  (see Equation 1) and initializes a resulting relaxed plan cost by the cost of the computed RP excluding all projected actions:  $c^{\alpha_i} \leftarrow \sum_{a \in \pi^{\alpha_i+} \cap A_i} c(a)$ .
2. For each projected action  $a^{\alpha_i} \in \pi^{\alpha_i+} \setminus A_i^+$ , the initiator agent  $\alpha_i$  sends a request to the action's owner agent  $\alpha_j$ . Upon receiving,  $\alpha_j$  computes partial RP  $\pi^{\alpha_j+}$  as a solution of a problem  $\langle P_j, A_j^+ \cup \bigcup_{\alpha_k \in \mathcal{A}} \{b^{\alpha_j} | b \in A_k^+\}, s \cap P_j, \text{pre}(a^+)\rangle$  and sends the cost of  $\pi^{\alpha_j+} \cap A_j^+$  to the initiator agent as a reply:  $c^{\alpha_j} \leftarrow \sum_{a \in \pi^{\alpha_j+} \cap A_j^+} c(a)$ .
3. The agent  $\alpha_j$  may need to ask other agent(s)  $\alpha_k$  in the same manner, resulting in a distributed recursion summing up the costs:  $c^{\alpha_j} \leftarrow c^{\alpha_j} + c^{\alpha_k}$ .

4. The initiator agent then adds the received cost to the resulting RP cost:  
 $c^{\alpha_i} \leftarrow c^{\alpha_i} + c^{\alpha_j}.$

The most severe drawback of Lazy FF is that although only local actions of agents are counted, significant over-counting of actions often appears due to the distributed recursion and possibly repeated counting of actions already counted by the same agents.

In [5], the lazy approach was also applied to the principle of Exploration Queue itself. The basic process of building the distributed exploration queue is similar to the centralized version, but whenever a projection of some other agent's action should be applied (and its effect added to the queue), a request is sent to the owner of the action to obtain its true cost first. This follows the lazy principle as described in 2nd step of Lazy FF. The effect of the action is added to the distributed queue after the reply is received. When an agent is computing the reply, the agent may need to send requests as well, thus ending up with a distributed recursion as in 3rd step of Lazy FF. In order to effectively handle the recursion it is flattened so that all requests are sent by the initiator agent  $\alpha_i$  and the replies are supplied with the parameters allowing the initiator agent  $\alpha_i$  to request the agents  $(\alpha_j, \alpha_k, \dots)$  of the next calls. The Relaxed Plan is then extracted only locally. The resulting heuristic, Recursive Distributed FF (or rdFF) exhibited better performance than Lazy FF using (local) Exploration Queues for the partial RP computation, most probably due to the over-counting of actions in Lazy FF.

### 3.1.3. Set-Additive Lazy FF

In this article, we introduce a new technique how to handle the over-counting in Lazy FF. We take inspiration from the Set-Additive variation of the FF heuristic [13], where instead of cost of reaching a fact  $p$  in a planning problem  $\Pi = \langle P, A, I, G \rangle$ , each fact  $p$  is associated with a relaxed plan  $\pi_p^+$  solving a relaxed planning problem where  $p$  is the goal  $G = \{p\}$ . The overall relaxed plan  $\pi^+$  is then constructed by computing a set unions of the respective fact relaxed plans

$$\pi^+ = \bigcup_{p \in P} \pi_p^+, \quad (2)$$

which is possible as the order of the actions in a relaxed plan can be arbitrary and using any action more than once is redundant. The new heuristic is termed Set-Additive Lazy FF or SA Lazy FF.

The estimation of SA Lazy FF proceeds similarly as in Lazy FF using Exploration Queues. The difference is that the agents do not send the intermediate costs  $c^{\alpha_j}, c^{\alpha_k}, \dots$ , but the relaxed plans  $\pi^{\alpha_j+}, \pi^{\alpha_k+}, \dots$  which are then merged by the initiating agent  $\alpha_i$ . The resulting heuristic estimate  $c^{\alpha_i}$  is the cost of the merged relaxed plan. The SA Lazy FF estimator in the recursive form follows:

1. The agent  $\alpha_i$  initiating the estimation locally computes a projected relaxed plan  $\pi^{\alpha_i+}$  which is a solution of a relaxed projected problem  $\Pi^{\alpha_i+}$  (see Equation 1).



2. For each projected action  $a^{\alpha_i+} \in \pi^{\alpha_i+} \setminus A_i^+$ , the initiator agent  $\alpha_i$  sends a request to the action's owner agent  $\alpha_j$ . Upon receiving,  $\alpha_j$  computes partial RP  $\pi^{\alpha_j}$  as a solution of a projected problem

$$\Pi^{\alpha_j+} = \left\langle P_j^{\text{proj}}, \bigcup_{\alpha_k \in \mathcal{A}} \{b^{\alpha_k+} | b \in A_k^+\}, s \cap P_j^{\text{proj}}, \text{pre}(a^+) \right\rangle$$

and sends  $\pi^{\alpha_j}$  to the initiator agent as a reply.

3. The agent  $\alpha_j$  may need to ask other agent(s)  $\alpha_k$  in the same manner, resulting in a distributed recursion merging the partial relaxed plans:  $\pi^{\alpha_j+} \leftarrow \pi^{\alpha_k+} \cup \pi^{\alpha_j+}$ .
4. The initiator agent merges the received relaxed plan  $\pi^{\alpha_j+}$  with the initial RP  $\pi^{\alpha_i+} \leftarrow \pi^{\alpha_i+} \cup \pi^{\alpha_j+}$  and returns its cost:  $\sum_{a \in \pi^{\alpha_i+}} c(a)$ .

Similarly to the Lazy FF estimator, we have to explicitly consider that in the 2nd step, no solution for  $\Pi^{\alpha_j+}, \Pi^{\alpha_k+}, \dots$  may exist. Such situation indicates a dead-end on preconditions  $\text{pre}(a)$ . As it is not cheaply possible to find out if another relaxed plan  $\pi^{\alpha_i+}$  could be used, we ignore the information and return the heuristic as if the action was reachable (and the replied relaxed plan  $\pi^{\alpha_j+}, \pi^{\alpha_k+}, \dots$  empty). This causes the heuristic to report non- $\infty$  estimates for dead-end states (will not be *safe*). Since the FF heuristic is not safe by itself and the practical efficiency gains are substantial, we conclude it is not a (practical) problem.

Following the same arguments as in the case of Lazy FF, for implementation of SA Lazy FF, we have flattened the recursion<sup>2</sup> and used the Exploration Queues ending with algorithm implementing the Equation 2 using distributed requests for  $\pi_p^+$  plans.

*Soundness..* In the SA Lazy FF heuristics we trade the estimation equality with centralized FF for efficiency of the distributed computation. Technically, each agent may choose different supporter action for a single fact, thus in the resulting relaxed plan a single fact may have multiple supporting actions. This unwanted effect causing overcounting of actions can be partially reduced by the following modification. When an agent receives a request, it computes relaxed plan only to the private preconditions of the action - the public preconditions were already taken care of by the initiator agent. Proof of soundness of the resulting relaxed plan  $\pi^{\alpha_i+}$  follows:

**Proposition 10.** *Let  $\Pi$  be a multiagent planning problem,  $\Pi^G$  the respective global problem and  $\Pi^{G+} = \langle P, A^+, I, G \rangle$  the respective relaxed global problem. A relaxed plan  $\pi^{\alpha_i+}$  computed by agent  $\alpha_i$  using the SA Lazy FF estimator is a sound solution of  $\Pi^{G+}$ .*

---

<sup>2</sup>For applications requiring “real” private knowledge preservation as proposed in [14], SA Lazy FF can use hashes as a way of obfuscation (and also communication load reduction) for the communicated private actions. It is questionable, whether some useful information can be extracted from the obfuscated relaxed plans, but if so, this approach may not be suitable for applications requiring such degree of privacy protection.

*Proof.* In the SA Lazy FF estimator, we start with a projected relaxed plan  $\pi^{\alpha_i+}$ , which is not a valid plan for the global relaxed problem  $\Pi^{G+}$ , because it may contain projected actions which are not part of the global problem. We can transform  $\pi^{\alpha_i+}$  by replacing all projected relaxed actions  $a^{\alpha_i+}$  by the original relaxed actions  $a^+$ . We will denote the resulting relaxed plan as  $\pi^{\alpha_i \rightarrow G}$ . Notice  $\pi^{\alpha_i \rightarrow G}$  is still not a valid plan for  $\Pi^{G+}$ , because it may contain actions which have some preconditions not met. Those are private preconditions of the originally projected actions. There are no actions to support them.

The SA Lazy FF estimator then continues with sending requests for each  $a^{\alpha_i+} \in \pi^{\alpha_i+} \setminus A_i^+$ . Neither the order of actions or the exact process of obtaining reply is important here. The reply obtained from some agent  $\alpha_j$  for action  $a^{\alpha_i+}$  s.t.  $a^+ \in A_j^+$  is a relaxed plan  $\pi^{\alpha_j+}$ , such that its application to the global initial state  $I$  results in satisfying all private preconditions of  $a^+$  (those are  $\text{pre}(a^+) \cap \text{facts}(\alpha_j)$ ). Of course,  $\pi^{\alpha_j+}$  may again contain projected actions of agents other than  $\alpha_j$  (including  $\alpha_i$ ).

Since by application of  $\pi^{\alpha_i+} \setminus a^{\alpha_i+}$  to the initial state  $I$ , all public preconditions of  $a^+$  are satisfied, union of the relaxed plans  $\pi^{\alpha_i, \alpha_j} = \pi^{\alpha_i} \cup \pi^{\alpha_j}$  satisfies all preconditions of  $a^+$ . The union may introduce new projected actions, but as there is a finite number of actions in the problem, all projected actions are eventually resolved by repeated application of the request-reply protocol. Potentially, there may be cycles in the partial relaxed plans (e.g. action  $a \in A_i$  achieves precondition  $p_b$  of action  $b \in A_j$  and  $b$  achieves precondition  $p_a$  of  $a$ ), but as the preconditions  $p_a, p_b$  are shared among the agents  $i, j$  they must be public and therefore either  $p_a$  or  $p_b$  must already be resolved in the RP.

When the protocol finishes, all actions in the resulting merged relaxed plan  $\pi^A$  have all their preconditions supported. Since already  $\pi^{\alpha_i}$  achieved the goal projected to agent  $\alpha_i$ , that is  $G^{\alpha_i}$  and since global goal is assumed, that is  $G^{\alpha_i} = G$ , also  $\pi^A$  achieves the global goal  $G$  and is a valid plan for the relaxed problem  $\Pi^{G+}$ .  $\square$

*MADLA Search compliance..* For the MADLA Search, by Definition 7 we need two heuristics  $h_L$  and  $h_D$ . As a local  $h_L$ , we use projected FF (see Section 3.1.1) and as a distributed (global)  $h_D$  we use SA Lazy FF defined in the previous paragraphs. Now, we can show the following:

**Proposition 11.** *The local FF heuristic  $h_L$  and the global SA Lazy FF heuristic  $h_D$  comply with the definition of MADLA heuristic pair (Definition 7).*

*Proof.* First,  $h_D$  dominates  $h_L$  (Definition 5): The Exploration Queue algorithm is in both cases the same, also the initial RP extraction procedure is the same. In addition SA Lazy FF sends requests and possibly augments the RP with received actions of other agents. Therefore the length (and thus the heuristic estimate) of the RP constructed by SA Lazy FF is always equal or longer than the one constructed by projected FF.

Second,  $h_D$  estimator is computationally harder than  $h_L$  estimator (Definition 6): The initial step of both heuristic evaluations is the same—building a local reachability analysis and finding the local relaxed plan. The SA Lazy FF

heuristic then continues with additional computation (also involving communication) to obtain the global RP estimate. Therefore the computation of SA Lazy FF always takes the same or longer time to finish, regardless the speed of the communication.

Last,  $h_D$  estimator is non-blocking (Definition 4): When the reachability is done and local RP extracted, SA Lazy FF sends requests to all agents whose projected actions were used in the RP. After sending the requests, the agent waits for the replies - while waiting, some other asynchronous computation can be performed. The agent must also compute replies to requests from other agents, but this does not break the property.  $\square$

To sum up the situation, we have two heuristic estimators—one less informative but fast and one more informative but slower. This is quite common situation in classical planning. In our case, the slower heuristic  $h_D$  is asynchronous (that is non-blocking), meaning the agent can perform other computation while it is being evaluated. The computation, the agents will perform meanwhile, is a local search driven by estimation of local  $h_L$  heuristics. As soon as the agent receives the estimation from  $h_D$  it can progress with the global search.

### 3.2. Search

In order to find a plan, the MADLA planner searches through a state-space induced by the input planning problem, driven by the proposed heuristic pair of local FF and distributed SA Lazy FF. From the literature [5, 15], we know that the performance of local and distributed heuristics differs over various multi-agent planning problems. The search we are proposing combines the local and distributed heuristics in a manner inspired by classical multi-heuristic search [9], but modified to be suitable for similar (or as in our case the same) heuristics in local and distributed forms.

Since MADLA is a multiagent planner, the search is a distributed computation spread over the agents. We build on the idea of distributed state-space search [3, 5] and extend it to a distributed multi-heuristic search.

Each building block is defined formally and instantiated for use with FF and SA Lazy FF heuristics. Similarly, the merged multiagent multi-heuristic search is defined generally and grounded into the particular MADLA Search with FF and SA Lazy FF. The MADLA search algorithm is then described in detail.

#### 3.2.1. Classical Multi-Heuristic Search

Multi-heuristic search was pioneered by the Fast Downward planning system [9] as a way how to combine different heuristic estimators without the need to combine the heuristic values, and was also one of the main mechanisms behind the success of the LAMA planner [16].

**Definition 12.** Let  $\Pi = \langle P, A, I, G \rangle$  be a planning problem and  $h_1, \dots, h_m$  heuristics functions for  $\Pi$ . A *multi-heuristic search* is defined over states  $s \in 2^P$  and uses *OPEN lists*  $O_1, \dots, O_m$  of states, a *CLOSED list*  $C$  of states, a selection function

$$\text{select}(O_1, \dots, O_m) \mapsto O_k$$

where  $1 \leq k \leq m$  and a state expansion function

$$\text{expand}(s') \mapsto \{s \mid s = \text{apply}(s', a), a \in A \text{ s.t. } \text{pre}(a) \subseteq s'\}.$$

A multi-heuristics search systematically extracts a state  $s = \arg \min_{s \in O_k} h_k(s)$  from  $O_k$  selected by the **select** function, adds  $s$  into  $C$  and adds states  $\text{expand}(s) \setminus C$  into  $O_i$ , for all  $1 \leq i \leq m$ . The search begins with  $O_i = \{I\}$  for all  $1 \leq i \leq m$  and terminates if  $G \subseteq s$ . The result is a sequence  $\pi = (a_1, \dots, a_k)$  of actions used in the expands from  $I$  to  $G$ .

To fit the heuristics pair to a multi-heuristic search, the local FF  $h_L$  and distributed SA Lazy FF  $h_D$  can be used in the following way:  $h_1 = h_L$ ,  $h_2 = h_D$  and  $m = 2$ . Thus, by Definition 12, there will be an OPEN list  $O_1 = O_L$  for states evaluated by the local heuristics and an OPEN list  $O_2 = O_D$  for the distributed heuristics.

A choice of an appropriate selection function **select** for classical planning was thoroughly examined in [10] with a conclusion that the simple alternation mechanism, where the open lists are chosen in turn, appears to be the best one (this mechanism was used in both FD and LAMA).

### 3.2.2. Multiagent Single-Heuristic Search

A multiagent variant of the heuristic search was proposed in [3, 5] as a multiagent optimal search and multiagent greedy best-first search (MA-BFS) respectively. The search in MADLA adheres to the latter.

The main principle of multiagent heuristic search is a straightforward distribution of the classical heuristic search as illustrated in Figure 1.

**Definition 13.** Let  $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G \rangle$  be a multiagent planning problem,  $h$  a heuristic function for  $\Pi$ . A *multiagent heuristic search* is defined over states  $s \in 2^P$  and over each agent's OPEN list  $O_{\alpha_i}$  of states, CLOSED list  $C_{\alpha_i}$  of states, agent state expansion function

$$\text{expand}_{\alpha_i}(s) \mapsto \{s' \mid s' = \text{apply}(s, a), a \in A_i \text{ s.t. } \text{pre}(a) \subseteq s\},$$

and a distribution function

$$\text{dist}_{\alpha_i} : 2^P \times A \rightarrow 2^A.$$

In *multiagent heuristic search*, each agent  $\alpha_i$  extracts systematically and in parallel a state  $s = \arg \min_{s \in O_{\alpha_i}} h(s)$  from  $O_{\alpha_i}$ , adds  $s$  into  $C_{\alpha_i}$  and expands states  $S = \text{expand}_{\alpha_i}(s) \setminus C_{\alpha_i}$ , where  $s' \in S$  are added into OPEN lists  $\{O_{\alpha_j} \mid \alpha_j \in \text{dist}_{\alpha_i}(s, a)\}$ , where action  $a$  expanded  $s'$ . The search begins with  $O_{\alpha_i} = \{I\}$  for all agents and terminates if any agent finds  $s$  s.t.  $G \subseteq s$ . The result is a sequence  $\pi = (a_1, \dots, a_k)$  of actions used in the expands from  $I$  to  $G$  across the agents.

MA-BFS [5] is a multiagent heuristic search by Definition 13. The used state exchange function **dist** is based on MA-STRIPS partitioning of the agent's

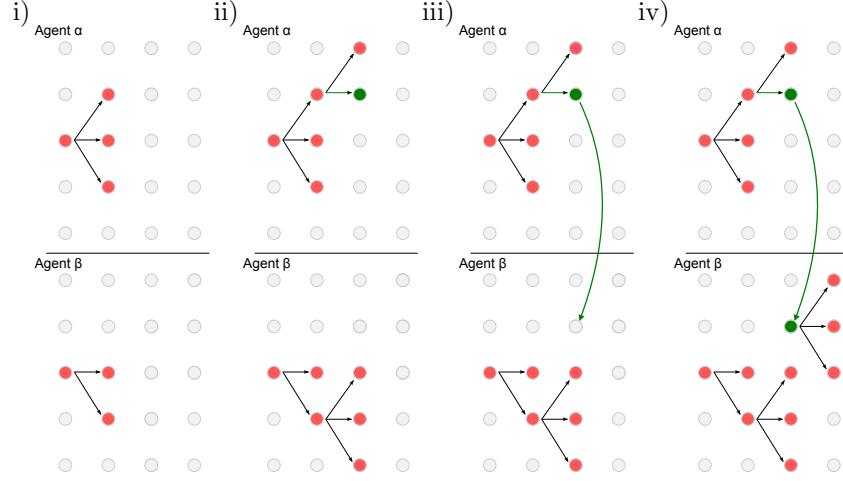


Figure 1: Example of a multiagent heuristic search for two agents  $\alpha$  and  $\beta$ . In i) both agents expand the initial state by private actions (black arrows), in ii) agent  $\alpha$  expands one state by a public action (green arrow and circle) and in iii) it is sent to agent  $\beta$ . In iv) agent  $\alpha$  have no longer an applicable action, but agent  $\beta$  expands the received state.

public actions  $A_i^{\text{pub}}$  and internal actions (as exemplified in Figure 1). From perspective of each agent  $\alpha_i$ , the MA-BFS distribution function is defined as

$$\text{dist}_{\alpha_i}^{\text{MA-BFS}}(a) \mapsto \begin{cases} \mathcal{A} & a \in A_i^{\text{pub}}, \\ \{\alpha_i\} & \text{otherwise.} \end{cases}$$

In MA-BFS implementation, adding expanded state  $s'$  to other agents OPEN lists uses message broadcast<sup>3</sup> initiated by the agent  $\alpha_i$ . The broadcast is represented in  $\text{dist}_{\alpha_i}^{\text{MA-BFS}}$  by the  $a \in A_i^{\text{pub}}$  case. The other case describes expand by an internal action of the agent  $\alpha_i$ , therefore the expanded state is added only into the OPEN list of the agent  $\alpha_i$ , following the principle of classical heuristic search.

### 3.2.3. Multiagent Multi-Heuristic Search

A merge of the two previous search schemes combines a multi-heuristic search (Definition 13) and a multiagent search (Definition 12)

**Definition 14.** Let  $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G \rangle$  be a multiagent planning problem and  $h_1, \dots, h_m$  heuristic functions for  $\Pi$ . A *multiagent multi-heuristic*

<sup>3</sup>As long as an agent  $\alpha_i$  does not want to share private facts  $P_i^s = \{p | p \in s \text{ s.t. } p \in P_i^{\text{int}}\}$  of a broadcasted state  $s$ , it can obfuscate the facts in  $P_i^s$  (as proposed in [14]) or replace the facts in set  $P_i^s$  by a private unique identifier (or a hash value) known only to  $\alpha_i$ , as this private part of the state cannot be modified by other agents. If a modified state amended by such identifier returns by one of the later broadcasts back to the agent  $\alpha_i$ , it can use the identifier and add the private facts  $P_i^s$  back as if they were always part of the state.

*search* is defined over states  $s \in 2^P$  and over each agent's  $\alpha_i$  OPEN lists  $O_{\langle\alpha_i,1\rangle}, \dots, O_{\langle\alpha_i,m\rangle}$  of states, CLOSED list  $C_{\alpha_i}$  of states, a selection function

$$\text{select}_{\alpha_i}(O_{\langle\alpha_i,1\rangle}, \dots, O_{\langle\alpha_i,m\rangle}) \mapsto O_{\langle\alpha_i,k\rangle}$$

where  $1 \leq k \leq m$ , a state expansion function

$$\text{expand}_{\alpha_i}(s') \mapsto \{s \mid s = \text{apply}(s', a), a \in A_i \text{ s.t. } \text{pre}(a) \subseteq s'\},$$

and a distribution function

$$\text{dist}_{\alpha_i} : 2^P \times A \rightarrow 2^{A \times \{1, \dots, m\}}.$$

In *multiagent multi-heuristic search*, each agent extracts systematically and in parallel a state  $s = \arg \min_{s \in O_{\langle\alpha_i,k\rangle}} h_k(s)$  from  $O_{\langle\alpha_i,k\rangle}$  selected by the  $\text{select}_{\alpha_i}$  function, adds  $s$  into  $C_{\alpha_i}$  and expands  $S = \text{expand}_{\alpha_i}(s) \setminus C_{\alpha_i}$ , where  $s' \in S$  are added into OPEN lists  $\{O_{\langle\alpha,k\rangle} \mid \langle\alpha,k\rangle \in \text{dist}_{\alpha_i}(s, a)\}$ , where  $s'$  was expanded by  $a$ . The search begins with  $O_{\langle\alpha,k\rangle} = \{I\}$  for all agents and all OPEN lists and terminates if any agent finds  $s$  s.t.  $G \subseteq s$ . The result is a sequence  $\pi = (a_1, \dots, a_l)$  of actions used in the expands from  $I$  to  $G$ .

As a baseline instantiation of the scheme we will use the alteration of local FF and SA Lazy FF OPEN lists and MA-BFS distribution function based on MA-STRIPS factorization. Such approach can be considered only a baseline, as the efficiency improvements of the multi-heuristic search (as e.g., in LAMA) was achieved thanks to the use of very different and in a sense “orthogonal” heuristics and cannot be anticipated when two variants of the same FF heuristic are used. However, the idea of an OPEN list for each heuristic can be modified to bring improvement in multi-agent heuristic search, as we will show in further sections.

This schema is also a template for the MADLA Search which is its instantiation with the local FF and SA Lazy FF heuristics and a specific selection and distribution functions described in the following section.

### 3.2.4. MADLA Search: Multiagent Distributed and Local Asynchronous Search

The search used in the MADLA planner is a multiagent multi-heuristic search driven by a distributed and a local heuristic. As we explain in the following paragraphs, the MADLA Search relies on the properties of the MADLA heuristic pair described in Definition 7, which were shown to hold for the local FF heuristic and the SA Lazy FF heuristic in Proposition 11.

Similarly, as we have shown for multi-heuristic search by Definition 13, the search schema is instantiated such that  $h_1 = h_L$ ,  $h_2 = h_D$  and  $m = 2$ , where  $h_L$  is the local FF and  $h_D$  the SA Lazy FF heuristic. Following the Definition 14, the search use two OPEN lists  $O_{\langle\alpha_i,L\rangle}, O_{\langle\alpha_i,D\rangle}$  for each agent  $\alpha_i$ . The selection function prioritizes expansion of states in the OPEN list related to the distributed heuristic, if the heuristic is not in the process of computing an heuristic estimate (denoted as  $\text{busy}_D$ ). In the case of SA Lazy FF heuristic,  $\text{busy}_D$  means that the heuristic is waiting for some replies. The MADLA selection function:

$$\text{select}_{\alpha_i}^{\text{MADLA}}(O_{\langle\alpha_i, \text{L}\rangle}, O_{\langle\alpha_i, \text{D}\rangle}) \mapsto \begin{cases} O_{\langle\alpha_i, \text{D}\rangle} & \neg\text{busy}_{\text{D}} \wedge O_{h_{\text{G}}h_{\text{G}}\langle\alpha_i, \text{D}\rangle} \neq \emptyset, \\ O_{\langle\alpha_i, \text{L}\rangle} & \neg\text{busy}_{\text{D}} \wedge O_{\langle\alpha_i, \text{D}\rangle} = \emptyset, \\ O_{\langle\alpha_i, \text{L}\rangle} & \text{busy}_{\text{D}}. \end{cases}$$

The main loop of the MADLA Search is listed in Algorithm 1. The algorithm follows the search by Definition 14.

Unlike the classical multi-heuristic search (Section 3.2.1), in MADLA search, the extracted states are not evaluated by both heuristics, the used heuristic evaluator depends instead on the state of the SA Lazy FF heuristic  $h_{\text{D}}$ . If  $\neg\text{busy}_{\text{D}}$  state is evaluated by  $h_{\text{D}}$ . If  $\text{busy}_{\text{D}}$ , state is evaluated by  $h_{\text{L}}$ , that is the distributed heuristic is always preferred. This approach is most reasonable if  $h_{\text{D}}$  dominates  $h_{\text{L}}$ , which holds for the local FF and SA Lazy FF heuristics (Proposition 11).

Additionally, the local heuristic search is performed only when the distributed heuristic search is waiting for the distributed heuristic estimation to finish (see line 5 in Algorithm 1). This principle makes sense only if finishing a estimation of  $h_{\text{D}}$  takes longer than of  $h_{\text{L}}$  and if computation of the  $h_{\text{D}}$  estimator does not block the search process (incl.  $h_{\text{L}}$  estimations). These two requirement hold for the local FF and SA Lazy FF by Proposition 11 as well.

Separation of the searches (Algorithms 2, 3, and 4) has the benefit of using two heuristics in parallel, but if some information between the two searches could be shared<sup>4</sup>, most importantly the heuristically best state found so far, it could boost the efficiency of the planner. The direction  $O_{\langle\alpha_i, \text{D}\rangle} \rightarrow O_{\langle\alpha_i, \text{L}\rangle}$  is straightforward, thanks to the fact that  $h_{\text{D}}$  dominates  $h_{\text{L}}$ . We can add all nodes evaluated by  $h_{\text{D}}$  also to the  $O_{\langle\alpha_i, \text{L}\rangle}$  without ever skipping a better state evaluated by  $h_{\text{L}}$  with a worse state evaluated by  $h_{\text{D}}$ .

The state distribution function covers this direction:

$$\text{dist}_{\alpha_i}^{\text{MADLA}}(s, a) \mapsto \begin{cases} \mathcal{A} \times \text{L} & a \in A_i^{\text{pub}} \wedge s \in O_{\langle\alpha_i, \text{L}\rangle}, \\ \{\langle\alpha_i, \text{L}\rangle\} & a \in A_i^{\text{int}} \wedge s \in O_{\langle\alpha_i, \text{L}\rangle}, \\ \mathcal{A} \times \text{D} \cup \mathcal{A} \times \text{L} & a \in A_i^{\text{pub}} \wedge s \in O_{\langle\alpha_i, \text{D}\rangle}, \\ \{\langle\alpha_i, \text{D}\rangle, \langle\alpha_i, \text{L}\rangle\} & \text{otherwise.} \end{cases}$$

The other direction  $O_{\langle\alpha_i, \text{L}\rangle} \rightarrow O_{\langle\alpha_i, \text{D}\rangle}$  is trickier. If we added a node  $s$  evaluated by  $h_{\text{L}}$  to  $O_{\langle\alpha_i, \text{D}\rangle}$  it would skip many nodes which are actually closer to the goal only because the local, less informative heuristic, will give a lower estimate. The way at least some information can be shared in this direction is whenever the OPEN list  $O_{\langle\alpha_i, \text{D}\rangle}$  becomes empty and the heuristic estimator  $h_{\text{D}}$  is not computing any heuristic, a state  $s$  is pulled from the local OPEN list  $O_{\langle\alpha_i, \text{L}\rangle}$  and evaluated by the distributed heuristic  $h_{\text{D}}$  and its successors are

---

<sup>4</sup>The two searches both run on one agent, therefore the question of privacy is irrelevant here.

---

**Algorithm 1:** MADLA Search for agent  $\alpha_i$  and multiagent planning problem  $\Pi$  following [Definition 14 of a multiagent multi-heuristic search](#).

---

```

1 Algorithm MADLA-Search
2    $O_{\langle\alpha_i, D\rangle} \leftarrow \{I\}; O_{\langle\alpha_i, L\rangle} \leftarrow \{I\}; C_{\alpha_i} \leftarrow \emptyset; //$  search begin
3   while solution not found do
4     processComm(); // see Algorithm 4
5     if  $\neg \text{busy}_D$  then
6       // select $^{\text{MADLA}}_{\alpha_i}$ 
7       if  $O_{\langle\alpha_i, D\rangle} \neq \emptyset$  then
8          $s \leftarrow \arg \min_{s \in O_{\langle\alpha_i, G\rangle}} h_D(s);$ 
9       else if  $O_{\langle\alpha_i, L\rangle} \neq \emptyset$  then
10         $s \leftarrow \arg \min_{s \in O_{\langle\alpha_i, L\rangle}} h_L(s);$ 
11      else
12        continue;
13      processState( $s, \text{true}$ );
14      while  $(O_{\langle\alpha_i, G\rangle} = \emptyset \text{ or } \text{busy}_D)$  and  $O_{\langle\alpha_i, L\rangle} \neq \emptyset$  do
15         $s \leftarrow \arg \min_{s \in O_{\langle\alpha_i, L\rangle}} h_L(s);$ 
16        processState( $s, \text{false}$ );
17        processComm(); // see Algorithm 4
18 Procedure processState( $s, d$ )
19   if  $s \notin C_{\alpha_i}$  then
20      $C_{\alpha_i} \leftarrow C_{\alpha_i} \cup \{s\}; //$  close state
21     if  $G \subseteq s$  then
22       reconstructPlan( $s, \emptyset$ ); // search end, see Algorithm 5
23     if  $d = \text{true}$  then
24       expandDistributed( $s$ ); // see Algorithm 2
25     else
26       expandLocal( $s$ ); // see Algorithm 3

```

---



---

**Algorithm 2:** expandLocal( $s$ )

---

```

1 Procedure expandLocal( $s$ )
2    $h_s \leftarrow h_L(s);$ 
3   if  $s$  reached by public action then
4     send STATE( $s, d = \text{false}$ );
5    $E \leftarrow \text{expand}(s); //$  expand $_{\alpha_i}$ 
6   // dist $^{\text{MADLA}}_{\alpha_i}$  two
7    $O_L \leftarrow O_L \cup E;$ 

```

---



---

**Algorithm 3:** expandDistributed( $s$ )

---

```
1 Procedure expandDistributed( $s$ )
2    $h_D(s, \text{callback}(h));$ 
3 handleCallback  $\text{callback}(h)$ 
4    $h_s \leftarrow h$ 
5   if  $s$  reached by public action then
6      $\text{send } STATE(s, d = \text{true});$ 
7    $E \leftarrow \text{expand}(s);$  //  $\text{expand}_{\alpha_i}$ 
8   //  $\text{dist}_{\alpha_i}^{\text{MADLA}}$ 
9    $O_D \leftarrow O_D \cup E;$ 
10   $O_L \leftarrow O_L \cup E;$ 
```

---

---

**Algorithm 4:** processComm()

---

```
1 Procedure processComm()
2   for each message  $M$  in message queue do
3     switch  $M$  do
4       //  $\text{dist}_{\alpha_i}^{\text{MADLA}}$ 
5       case  $M = STATE(s, \text{false})$ 
6          $O_L \leftarrow O_L \cup \{s\};$ 
7       case  $M = STATE(s, \text{true})$ 
8          $O_D \leftarrow O_D \cup \{s\};$ 
9          $O_L \leftarrow O_L \cup \{s\};$ 
10      case  $M = RECONSTRUCT(s, P)$ 
11         $\text{reconstructPlan}(s, P);$ 
```

---

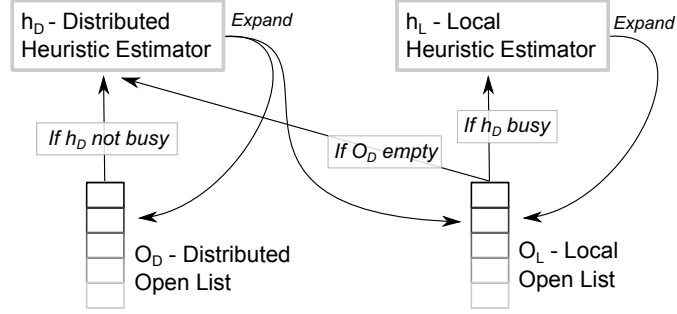


Figure 2: Distributed/Local Search - OPEN lists and heuristic estimators.

added to both OPEN lists, as already described. This way, the nodes in  $O_{\langle\alpha_i, D\rangle}$  are evaluated only by  $h_G$ , but sometimes, the best node from  $O_{\langle\alpha_i, L\rangle}$  is taken. The situation is illustrated in Figure 2.

The implementation of the distribution in the proposed search follows the principles of MA-BFS, i.e., broadcasts are used to inform other agents about states reached by public actions. Additionally, information to which OPEN list the state should be added in is included (whether it is the local or the distributed one). We use a *deferred* heuristic evaluation a technique used in LAMA planner for increasing efficiency of planning with computationally heavy heuristics.

In order to end with one sequence of actions  $\pi$  across all the agents by Definition 14, the reconstruction has to be a distributed process. In Algorithm 5 we list such algorithm and by that we close the description of the MADLA Search and the MADLA planner.

In the following section we prove that the resulting sequence  $\pi$  is a sound multiagent plan solving the input multiagent planning problem  $\Pi$  and that the MADLA planner is complete.

---

**Algorithm 5:** reconstructPlan( $s, P$ )

---

```

1 Procedure reconstructPlan( $s, P$ )
2    $P \leftarrow P \cup \{s\}$ 
3   if  $s = I$  then
4     solution found  $P$ ;
5   if parent( $s$ ) is local then
6     reconstructPlan(parent( $s$ ),  $P$ );
7   else
8     send RECONSTRUCT( $s, P$ ) to agent from which  $s$  was
      received

```

---

### 3.3. Soundness and Completeness

Before presenting the proof itself, the necessary assumptions will be stated:

**Assumption 1.** Liveness of the communication, i.e, every message will eventually arrive to its destination.

**Assumption 2.** Distributed heuristic is independent of the search, i.e, it is asynchronous and the messages never interfere.

**Assumption 3.** Distributed heuristic always terminates, i.e, for each state, distributed heuristic will eventually return the heuristic estimate.

Moreover, in the first part of the proof, we will assume that the search is performed by a single agent (although the heuristic still maintains assumptions 2, and 3.) on a classical STRIPS problem. The single-agent version of the algorithm will be denoted as SADLA Search. There is no communication in SADLA Search, therefore the `processComm()` procedure is ignored as well as all `send` commands. Later on, the proof will be extended to include multiple agents.

#### 3.3.1. Soundness

To show soundness of the SADLA Search, we will use the fact, that for each search state  $s$  we know its direct predecessor  $\text{parent}(s)$ . Using procedure `reconstructPlan()` shown in Algorithm 5, the sequence of states  $(s_0, s_1, \dots, s_k)$ , where for each  $0 < i \leq k$ ,  $s_{i-1} = \text{parent}(s_i)$ ,  $s_0 = I$  and  $s_k = s$ , can always be reconstructed. The action  $a_i$  which transformed state  $s_{i-1}$  to state  $s_i$  will be denoted as  $a_i = \text{trans}(s_{i-1}, s_i)$ .

**Definition 15.** A path is a sequence of states  $(s_0, s_1, \dots, s_k)$ . A path is valid iff  $s_0 = I$  is the initial state and for each  $0 < i \leq k$ :  $s_{i-1} = \text{parent}(s_i)$  and  $a_i = \text{trans}(s_{i-1}, s_i)$  is applicable in  $s_{i-1}$ . We say, that  $s_k$  is the resulting state and  $\text{path}(s_k)$  is the respective path (the sequence of states expanded during the search). The path is a valid solution, if  $s_k$  is a goal state, that is  $G \subseteq s_k$ .

To prove the soundness, the following lemma will be shown first:

**Lemma 16. Invariant:** *At any given step of the SADLA Search, for any state  $s_L \in O_L$  and any state  $s_D \in O_D$ ,  $\text{path}(s_L)$  and  $\text{path}(s_D)$  are valid paths.*

*Proof.* In the initial step,  $O_L = O_D = \{I\}$ , where  $\text{path}(I) = (I)$ , which is a valid path. The only procedures, where new states are added to any of the open lists are `expandLocal(s)` and `expandDistributed(s)` (note, that in SADLA Search, procedure `processComm()` is ignored). In both procedures, the states added are created using `expand(s)`, which creates new state  $s'$  for each action  $a$  applicable in  $s$  such that  $a = \text{trans}(s, s')$  and  $s = \text{parent}(s')$ . If we assume, that  $\text{path}(s) = (I, \dots, s)$  is a valid path, then after expansion,  $\text{path}(s') = (I, \dots, s, s')$  is also valid for each new  $s'$ .  $\square$

**Theorem 17.** *When the SADLA Search terminates and returns a solution, it is a valid solution.*

*Proof.* The algorithm terminates on line 4 of Algorithm 5. Since we assume a single agent, the procedure is a simple recursion and thanks to the condition on line 3 of Algorithm 5, upon termination, the last state added to the returned solution  $P$  is the initial state  $I$ . Procedure `reconstructPlan()` is initially called only from line 22 of Algorithm 1, where  $s$  was extracted from  $O_L$  or  $O_D$ . From Lemma 16 and because a state  $s'$  is added to  $P$  only if  $s = \text{parent}(s')$  follows that  $P$  is a valid path, i.e  $P = \text{path}(s)$ . Also, on line 22 of Algorithm 1 always holds that  $G \subseteq s$ , therefore  $P = \text{path}(s)$  is a valid solution.  $\square$

### 3.3.2. Completeness

To show completeness, the algorithm will be further modified, in order to show, that any reachable state can be reached by the algorithm. The modified algorithm will be denoted SADLA<sup>+</sup> Search in which the `reconstructPlan()` procedure in Algorithm 5 will be ignored and the algorithm will be terminated when both  $O_L$  and  $O_D$  are empty instead.

**Lemma 18.** *Each state is added to  $O_D$  and to  $O_L$  at most finite times.*

*Proof.* Because the number of possible states is finite ( $2^{|P|}$  since  $s \subseteq P$ ) and the number of actions is also finite, each call to `expand()` produces a finite number of states consequently added to  $O_D$ ,  $O_L$  or both (line 5 of Algorithm 2 and line 7 of Algorithm 3). If a state is extracted from  $O_L$  or  $O_D$ , it is added to the closed list  $C$  and never added to any of the open lists again.  $\square$

**Lemma 19.** *Each state in  $O_L$  and each state in  $O_D$  is eventually extracted.*

*Proof.* In each step of the outer while cycle of Algorithm 1, a state is extracted from  $O_D$ , if  $h_D$  is not busy. Since  $h_D$  always terminates (Assumption 3), the number of extracted states is not limited. From Lemma 18 follows, that only a finite number of states may be added to  $O_D$ , therefore  $O_D$  eventually becomes empty. When  $O_D$  is empty, in each step of the inner while cycle a state is extracted from  $O_L$ . Following the same reasoning as before.  $O_L$  eventually becomes empty as well.  $\square$

**Theorem 20.** *The SADLA<sup>+</sup> Search terminates.*

*Proof.* Follows directly from Lemma 18 and Lemma 19.  $\square$

**Definition 21.** A state  $s$  is reachable, if valid path  $\text{path}(s) = (I, \dots, s)$  exists.

**Lemma 22.** *If a state  $s$  is reachable, it is placed into the closed list  $C$  after a finite number of steps.*

*Proof.* Assume that  $s$  is reachable, but is never placed in  $C$ . Since  $s$  is reachable, then  $\text{path}(s) = (I, \dots, s)$ . Let  $s_i$  be the first state in  $\text{path}(s)$ , that is not added to  $C$ . Note, that there exist an action  $a$ , such that  $s_i = \text{apply}(a, s_{i-1})$ . Since at some point.  $s_{i-1} \in C$ ,  $s_{i-1}$  must have been taken from  $O_L$  or  $O_D$ . At that point.  $s_{i-1}$  was also expanded and because  $a$  is applicable in  $s_{i-1}$  it must have been applied. The resulting state  $s_i = \text{apply}(a, s_{i-1})$  was added to either  $O_L$  or  $O_D$  and because of Lemma 19 eventually extracted and added to  $C$ . This contradicts the assumption that  $s_i$  is not added to  $C$ .  $\square$

**Theorem 23.** *The SADLA Search is complete.*

*Proof.* In  $\text{SADLA}^+$  search, every reachable state  $s$  such that  $g \subseteq G$  is eventually placed into  $C$  (Lemma 22). Let  $s_0$  be first such state. In SADLA Search, after adding it to the closed list  $C$ , it is given to the Procedure 5 and  $\text{path}(s_0)$  is reported as a solution.  $\square$

### 3.3.3. Extending the Proof to Multiple Agents

The soundness and completeness shown for the SADLA Search holds also for the MADLA Search, that is, communicating states reached by a public action does not break the proven lemmas and theorems. Structures belonging to agent  $\alpha_i$  will be denoted by a superscript.

**Lemma 24.** *Sending state  $s$  from agent  $\alpha_i$  to agent  $\alpha_j$  for any  $1 \leq i, j \leq n$  does not violate the invariant of Lemma 16.*

*Proof.* If  $\text{path}(s)$  is valid before sending state  $s$  by agent  $\alpha_i$ , it is also valid after receiving state  $s$  by agent  $\alpha_j$  (no state is added or removed). When state  $s$  is sent by  $\alpha_i$ , the invariant holds, because one of the following holds:

- (i)  $\text{path}(s)$  contains no state previously received from another agent. Before sending  $s$ ,  $s' = \text{parent}(s)$  was extracted from either  $O_L^\alpha$  or  $O_D^\alpha$ , therefore from Lemma 16 follows that invariant holds for  $s'$ . By expanding a state, the invariant is not violated, therefore invariant holds for  $s$  as well.
- (ii)  $\text{path}(s)$  contains some states received from other agents. Let  $s'$  be first such state. Because of (i) and because sending does not violate the invariant, the invariant holds also for  $s$ .

$\square$

**Lemma 25.** *Procedure `reconstructPlan()` of Algorithm 5 reconstructs a valid plan even if multiple agents are involved.*

*Proof.* In each call of the procedure, either  $\text{parent}(s)$  is added (if it is local), or the  $\text{RECONSTRUCT}(s, P)$  message is sent. When received (each message is eventually delivered thanks to Assumption 1), the `reconstructPlan()` procedure is called again by the receiving agent. Since  $\text{path}(s)$  is valid, the process will eventually reach the initial state  $I$  and terminate.  $\square$

**Theorem 26.** *The MADLA Search is sound.*

*Proof.* Lemma 16 holds for MADLA Search because of Lemma 24. Soundness of MADLA Search follows from proof of soundness of SADLA Search (Theorem 17) and Lemma 25.  $\square$

**Lemma 27.** *Each state is received by any agent  $\alpha_i$  at most finite times.*

*Proof.* A state  $s$  is sent by agent  $\alpha_j$ , only if another state  $s'$  was extracted from either  $O_L^{\alpha_j}$  or  $O_D^{\alpha_j}$  and a public action  $a \in A_j^{\text{pub}}$  was applied. Since there is a finite number of public actions and a finite number of agents, each action is applicable only in finite number of states (which are then placed into  $C$  and never expanded again), therefore state  $s$  can be sent and received only finite number of times.  $\square$

Let  $\text{MADLA}^+$  Search denote a modification of MADLA Search such that the `reconstructPlan()` procedure in Algorithm 5 will be ignored and the algorithm will be terminated when  $O_L^{\alpha_i}$  and  $O_D^{\alpha_i}$  for all  $\alpha_i \in \mathcal{A}$  are empty.

**Lemma 28.** *The  $\text{MADLA}^+$  Search terminates.*

*Proof.* Lemma 18 holds also for  $\text{MADLA}^+$  Search, because the only additional situation in which a state is added to any open list is when it is received from another agent, which can happen at most finite times (Lemma 27). Therefore also Lemma 19 holds and the termination follows directly.  $\square$

In order to continue with the proof of completeness, we need to redefine the definition of reachability for multiple agents.

**Definition 29.** A state  $s$  is reachable by a sequence of agents  $\bar{\mathcal{A}} = (\alpha_0, \dots, \alpha_m)$  of length  $m$  (agents in  $\bar{\mathcal{A}}$  can repeat) if a sequence of states  $\bar{\pi} = (s_0^{\alpha_0}, \dots, s_{k_0}^{\alpha_0}, s_{k_0+1}^{\alpha_1}, \dots, s_{k_1}^{\alpha_1}, \dots, s_{k_m}^{\alpha_m})$  exists such that  $s_0^{\alpha_0}$  is the initial state  $I$  and was reached by agent  $\alpha_0$ ,  $s_{k_n}^{\alpha_m} = s$  and was reached by agent  $\alpha_m$  and  $\bar{\pi}$  is a valid path, i.e  $\bar{\pi} = \text{path}(s)$ .

**Lemma 30.** *If a state  $s$  is reachable by sequence of agents  $\bar{\mathcal{A}} = (\alpha, \dots, \alpha_k)$  of length  $k$ , it is placed into the closed list  $C$  after a finite number of steps.*

*Proof.* If a state  $s$  is reachable by a sequence containing single agent  $\bar{\mathcal{A}} = (\alpha_0)$ , the lemma holds trivially. Let us now assume, that for all  $k \leq m$  if a state is reachable by a sequence of agents  $\bar{\mathcal{A}} = (\alpha, \dots, \alpha_k)$  of length  $k$ , it is added to  $C$  after finite many steps. We will show, that the same holds if a state is reachable by sequence of agents  $\bar{\mathcal{A}}' = (\alpha_0, \dots, \alpha_m, \alpha_{m+1})$  of length  $m + 1$ . We will show the induction step by a contradiction. For the contradiction let us assume, that  $s$  is a state reachable by sequence of agents  $\bar{\mathcal{A}}'$  of length  $m + 1$ , but never added to  $C$ . Let  $\text{path}(s) = (s_0, \dots, s)$  and let  $s_i$  be the first state that is never added to  $C$ . One of the following holds:

- (i)  $s_i$  is reachable by agent  $\alpha_{m+1}$ . If so, the same reasoning used in the proof of Lemma 22 can be used to obtain a contradiction.

- (ii)  $s_i$  is reachable by some  $k$  agents  $\bar{\mathcal{A}} = (\alpha_0, \dots, \alpha_k)$  then we have a contradiction with the assumption of the induction. □

**Theorem 31.** *The MADLA Search is complete.*

*Proof.* Similarly to the single-agent version, in MADLA<sup>+</sup> search, every reachable state  $s$  such that  $s \subseteq G$  is eventually placed into  $C$  (Lemma 30). Let  $s_0$  be first such state. In MADLA Search, after adding it to the closed list  $C$ , it is given to the `reconstructPlan()` procedure which reports a valid solution  $P = \text{path}(s_0)$ , even if multiple agents are involved (Lemma 25). □

#### 4. Evaluation and Discussion

After the formal verification of the proposed MADLA search, we analyze its practical properties in experimental evaluation. First, we compare the building blocks of the planner, namely projected FF and distributed SA Lazy FF heuristics in multiagent single-heuristic search, the pair of the heuristics in multiagent multi-heuristic search, and finally the MADLA search. Second, we discuss in detail properties of the planner on particular planning domains and on various metrics. Finally, we compare the planner with other distributed multiagent planners on a multiagent benchmark set.

In the classical planning literature, the typical way how to compare planners or heuristics is to run them on a set of benchmark domains, each having a number of problem instances. The domains (or problems) vary in many properties, such as size, combinatorial hardness, structural traits and others. The same methodology was adopted in multiagent planning. A notable difference is in the set of benchmark domains, which is much less “standardized” in multiagent planning than in classical planning, as there is no track at International Planning Competition (IPC) [17] for multiagent planning yet. Therefore the set of benchmarks we are using in this article is a union of domains used for comparison of the four planners we are comparing MADLA to in the last section. All these domains and problems originate in single-agent domains and problems from IPC. Since the IPC domains are designed to both examine efficiency of the planners against various (theoretically) interesting properties and relate to real world problems, the same is anticipated in the case of multiagent planners. Currently there are not many properties specific only to the multiagent planners (as most of the multiagent planners are variations on distribution of principles used in classical planning), therefore no special multiagent domains and problems are usually used. The most notable property studied already by [2] is coupling of the agents in the problems which correlates (to some extent) with ratio of private and public actions in the planning problems. In the detailed analysis section, we will discuss influence of this and other more subtle traits of the problems influencing efficiency of used search and heuristics.

As the core comparison metrics, we will use coverage of solved problems under 20 minutes with 8GB memory limit. In the detailed analysis, we will use

number of expanded states (and their ratios for the used heuristics), computation time of the particular heuristics and number of public actions requiring a (private) action supporter for a private precondition fact.

The MADLA planner is a distributed multiagent system, therefore the non-determinism of the planning process is inherent and impossible to be synchronized under the assumption of unknown ordering in message delivery from two different agents to one recipient. Considering this assumption, every measurement was repeated 10 times<sup>5</sup> and the results were averaged. The experiments run on a homogeneous cluster of machines each dedicated for one run of the planner. Each machine was equipped with 8 hyper-threading i7 cores (i.e. 16 threads) at 2.6GHz. Each agent was running on two threads. One receiving messages and filling in content data structures in appropriate collections (e.g., states into the OPEN lists  $O_{\langle\alpha_i, L\rangle}$ ,  $O_{\langle\alpha_i, D\rangle}$  and the other searching and evaluating the heuristics.

#### 4.1. Comparison of the Building Blocks

The building blocks of the MADLA planner are the projected and distributed FF heuristics and the scheme how to combine these in search. A baseline approach (as we presented in Section 3.2.1) is to adapt classical multi-heuristic (MH) search for multiagent planning. However, this approach is not viable as the heuristics are not “orthogonal”. The proposed MADLA search utilizes the requirement for a non-blocking distributed heuristic estimator (particularly implemented in the form of the SA Lazy FF heuristic) by running projected FF in the spare time. The Table 1 summarizes the coverage of the FF and SA Lazy FF heuristics in separate multiagent single-heuristic search (Section 3.2.2), a multiagent multi-heuristic search (Section 3.2.3) and the MADLA search (Section 3.2.4) with the projected FF and distributed SA Lazy FF heuristics.

The results clearly indicate that the multi-heuristic approach is not suitable for the pair of projected and distributed FF heuristics. The summed up coverage results are similar for the single-heuristic searches following the results in our previous work [5]. The price for better estimates by the distributed variant of global FF is slower computation of the estimation. *Driverlog*, *elevators08*, *wood-working08* and *zenotravel* are domains where one of the heuristics is performing better. In cases where the FF heuristics is not appropriate or the overhead of its distributed computation overweights the fact the heuristics is more informative, the search using only projected FF performs better.

In MADLA search the distributed search is prioritized over the local projected search. Provided that the implementation is ideal, the coverage of the MADLA search should be always equal or better than the results of a single-heuristic search with the distributed SA Lazy FF heuristic. As the results

---

<sup>5</sup>Although 10 samples is not enough for a reasonable statistical confidence, it helps to identify cases with extreme variance. Such phenomenon did not manifest in any of our experiments, therefore we concluded the planner is deterministic enough for the used metrics. Deeper analysis of statistical properties of a multiagent (multi-heuristic) search is an interesting topic left for future work.



domain	$ \mathcal{A} $	FF	SA Lazy FF	MH search	MADLA search
blocksworld (35)	4	31.2	32.8	15	<b>34.2</b>
depot (20)	5-12	10.6	9.2	7	<b>12.4</b>
driverlog (20)	2-8	<b>17.3</b>	14	13	15.8
elevators08 (30)	4-5	17.3	28.1	2	<b>29.7</b>
logistics00 (20)	3-7	19.9	<b>20</b>	3	<b>20</b>
openstacks (30)	2	17.2	17.1	15.5	<b>17.5</b>
rovers (20)	1-8	<b>20</b>	19.9	6.6	<b>20</b>
satellites (20)	1-5	<b>20</b>	<b>20</b>	6	<b>20</b>
woodworking08 (30)	7	1.8	5	<b>5.6</b>	4.5
zenotravel (20)	1-5	19	14.1	8	<b>19</b>
total (245)		174.3	180.2	81.7	<b>192.9</b>

Table 1: Coverage of the building blocks. Number of problems in a domain are in the brackets,  $|\mathcal{A}|$  denotes the number (interval) of agents in the problems.

show, this property holds with one notable exception: the **woodworking08** domain. Since the current implementation does not interrupt the local estimation process of projected FF, the distributed estimator can be blocked for a time proportional to the hardness of the computation of the local FF. This phenomenon exhibits in (computationally hard) **woodworking08** problems and imply degradation even under the result of single-heuristic SA Lazy FF search. This is not the case for MH, as it estimated all states with both heuristics and alternates which OPEN list is used for expansion.

The only other case where the MADLA search is not over-performing both single-heuristic searches is the case of **driverlog**. In this case the explanation is straightforward. The prioritized SA Lazy FF heuristics is not appropriate for the particular problem and the projected version is faster because of its relative computational ease even though it is less informed.

With exception of **woodworking08** and **driverlog**, the proposed search scheme always improves the coverage over both single-heuristic searches and doubles the performance of the classical multi-heuristic scheme with the same heuristics.

#### 4.2. Detailed Analysis in Selected Domains

In this section, we analyze the performance of presented MADLA Search in detail. The Table 2 shows a comparison of various metrics measured using a standard Greedy Best-First Search (GBFS) with projected FF, distributed SA Lazy FF and MADLA search using both. The first two columns are ratios of coverage and the number of expanded states of GBFS with projected FF and distributed FF respectively. The next two columns shows the percentage of states in MADLA search expanded using projected FF and the ratio of states expanded using projected FF and distributed FF. The next three columns shows the time per state (in milliseconds), the MADLA search spends on computing the projected and distributed heuristics and the distributed/projected ratio. The last two columns show the average percentage of public and privately-dependent (PD) actions in the domain. We will not introduce the concept of PD

domain	$\frac{\text{projFF}}{\text{salFF}}$		MADLA exp		MADLA $t_h/s$			action ratio	
	cvg	exp	$\frac{\text{projFF}}{\text{all}}$	$\frac{\text{projFF}}{\text{salFF}}$	projFF	salFF	$\frac{\text{salFF}}{\text{projFF}}$	$\frac{\text{public}}{\text{all}}$	$\frac{\text{PD}}{\text{all}}$
<b>blocksworld</b>	<b>0.95</b>	<b>3.9</b>	<b>82</b>	<b>4.6</b>	<b>0.31</b>	<b>0.77</b>	<b>2.5</b>	<b>100</b>	<b>100</b>
<i>depot</i>	<i>1.15</i>	<i>0.8</i>	<i>92</i>	<i>11.5</i>	<i>0.65</i>	<i>4.76</i>	<i>7.3</i>	<i>95.7</i>	<i>23</i>
<i>driverlog</i>	<i>1.24</i>	<i>2.1</i>	<i>72</i>	<i>2.6</i>	<i>0.62</i>	<i>2.17</i>	<i>3.5</i>	<i>91.9</i>	<i>26.7</i>
<b>elevators08</b>	<b>0.62</b>	<b>42.6</b>	<b>81</b>	<b>4.3</b>	<b>0.36</b>	<b>0.57</b>	<b>1.6</b>	<b>66</b>	<b>66</b>
logistics00	1.00	29.7	89	8.1	0.16	0.63	3.9	67.4	33.7
openstacks	1.01	1.6	66	1.9	3.04	11.12	3.7	100	0
rovers	1.01	3.1	71	2.4	0.23	0.81	3.5	26.1	11.5
satellites	1.00	0.7	68	2.1	0.22	0.63	2.9	7.2	2.7
<b>woodwork.</b>	<b>0.36</b>	<b>11.9</b>	<b>65</b>	<b>1.9</b>	<b>1.07</b>	<b>2.51</b>	<b>2.3</b>	<b>99.9</b>	<b>13</b>
<i>zenotravel</i>	<i>1.35</i>	<i>0.2</i>	<i>69</i>	<i>2.2</i>	<i>0.53</i>	<i>0.58</i>	<i>1.1</i>	<i>20.7</i>	<i>14.1</i>

Table 2: Comparison of various metrics (coverage, expanded states, heuristic computation time, public and privately-dependent action ratios) using projected FF (projFF) and SA Lazy FF (salFF).

actions formally, but intuitively those can be seen as actions which are public, but have some private preconditions—this means, there is some dependence or causality hidden for agents other than the owner of the action.

A more detailed view on comparison of the projected and distributed FF heuristic in GBFS is presented in Figure 3. Left are the heuristic values for the initial state of all problems in selected domains for which the value was computed. It is clear that for most of the domains, as the complexity of the problem grows, also the difference between the distributed and projected heuristic grows (note, this does not say anything about the heuristic quality). Right is the number of expanded states (restricted to problems which were solved by at least one of the heuristics).

Together this two plots show some interesting properties. First, the **elevators08** domain is an example of domain, where the distributed heuristic gives much larger heuristic estimates, which also seems to be significantly more informed, as suggested by the number of expanded states. As the heuristic difference grows, also the difference of the number of expanded states grows in favor of the distributed heuristic. Similar behavior, only not as prominent, can be observed in the **blocksworld** domain. A completely different picture paints the **depot** domain, where the distributed heuristic also gives significantly larger estimates, but as shown in the expanded states plot, the heuristic guidance degrades and for larger problems, the projected heuristic is better for the search. The **driverlog** domain also fits into this category, where the larger distributed heuristic estimates does not necessarily lead the search better. On the other hand, in the **woodworking08** domain, we can observe that, although the heuristic estimates are pretty much the same for both heuristics, the number of states expanded by the projected heuristic grows in comparison with the distributed FF, which

suggests that even slight differences in the heuristic may have significant impact on the heuristic quality and its ability to lead the GBFS.

Now, we analyze the results shown in the Table 2 for each of the domains in detail.

**blocksworld** This domain is the same as the classical **blocksworld** domain except for having multiple hands as agents, the holding and free facts being private. Each agent can solve the problem on its own, but for the projected heuristic it seems that the solution by other agents is cheaper. This is also suggested by that all actions in the domain are privately-dependent (PD), this means there is some causality not known to the projected heuristic. This results in better heuristic guidance of the distributed heuristic, best-first search expanding almost  $4\times$  more states with projected heuristic than with the distributed one (column *exp* in Table 2). In MADLA search, there is about 20% of states expanded using distributed heuristic (column *MADLA exp* left), which seems to be enough to utilize the better heuristic guidance, but also with 80% of states expanded locally utilizing the  $2.5\times$  speedup per state (column *MADLA exp* left and *MADLA  $t_h/s$*  right), thus solving even more problems than GBFS with distributed FF.

**depot** In **depot**, the trucks, depots, and distributors are agents. Most of the actions are public, nearly quarter of them have private preconditions (PD actions), but those are actions which operate with non-shared locations, that is typically initial states, which diminishes the effect of PD actions. Completely private are only drive-truck actions. The distributed heuristic takes approx.  $7\times$  longer to evaluate in average (column *MADLA  $t_h/s$*  right in Table 2), but on its own, the heuristic guidance is rather poor (GBFS with distributed heuristic expanding over 20% more states, column *exp*). This leads to worse performance of search guided only by the distributed FF heuristic. In MADLA, due to the time demanding distributed heuristic computation, about 92% of states is expanded using the projected heuristic (column *MADLA exp* left). Nevertheless, the small number of states expanded using the distributed heuristic improves the result even in comparison with GBFS using only the projected FF.

**driverlog** In **driverlog**, the drivers are agents, their locations, walk action and the fact that a driver is driving a truck are private, everything else is public. Most of the problems can be solved by a single agent. The distributed heuristic seems to lead the search slightly better (approx.  $2\times$  less expanded states, column *exp* in Table 2), but takes approx.  $3.5\times$  more time per state which results in significantly worse coverage (column *MADLA  $t_h/s$*  right, column *cvg* and Table 1). In MADLA, this is partially improved by approx 70% of states expanded using the projection (column *MADLA exp* left), but it is not enough to reach the coverage score of the projected heuristic on its own.

**elevators08** In the **elevators** domain, the elevators are agents, locations of passengers are public only if shared among multiple elevators (changing

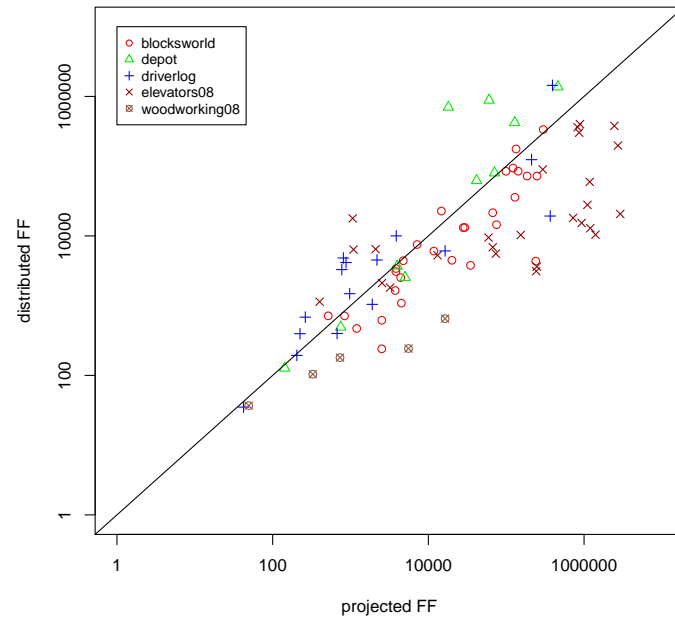
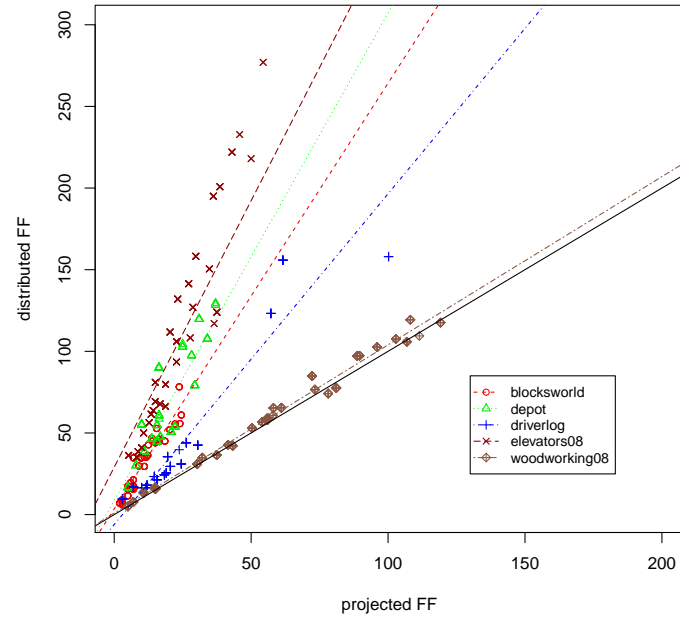


Figure 3: Heuristic values for initial states and number of expanded states on selected domains.

floors). Public actions are only those board and leave actions involving a shared floor. Most of the problems can be solved by a subset of agents (typically the slow elevators). All public actions have private preconditions on the state of the lift, its capacity, etc (they are privately-dependent) which makes the distributed heuristic dramatically more accurate. The GBFS with a projected heuristic expands over  $40\times$  more states than with the distributed one (column *exp* in Table 2) and the distributed heuristic takes in average only approx.  $1.5\times$  longer to compute (column *MADLA  $t_h/s$*  right). This results in over 10 problem difference in coverage in favor of the distributed heuristic (Table 1). In MADLA, about 20% of states is expanded using the global heuristic (column *MADLA exp* left) which is enough not only to match the performance of GBFS with distributed heuristic, but to improve it slightly.

**woodworking08** In the woodworking domain, each tool is an agent. All facts and actions in this domain are public, except for the fact stating that a high-speed saw is empty or loaded. Subsequently, loading and unloading the high-speed saw are the only public actions with private preconditions (privately-dependent). In GBFS the distributed heuristic expands approx.  $12\times$  less states and takes  $2.3\times$  longer to evaluate per state (columns *exp* and *MADLA  $t_h/s$*  right in Table 2), which result in solving more than twice as many problems using the distributed heuristic. In MADLA, however, the result is slightly worse. This is because the problem itself is hard and even the projected heuristic takes a substantial time to compute, as more than 60% of states is expanded using projected heuristic (column *MADLA exp* left), which does not provide as good heuristic guidance, the whole process is hindered and less problems solved. Note, that as the experiments are evaluated statistically, the difference of 0.5 problems is not significant and may as well be a result of the non-determinism in the computation.

Concerning other domains, **zenotravel** (agents as planes are transporting passengers) is much similar to **driverlog**, except the heuristic guidance of distributed heuristic is significantly worse than that of projected heuristic, but the distributed heuristic takes nearly the same amount of time to compute, which in MADLA eliminates its negative effect. Based on the measured values and also on the understanding of the domain, **logistics00** domain (trucks and planes as agents transporting packages) is similar to the **elevators08** domain, although much easier to solve. The **rovers** and **satellites** domains are very loosely coupled domains with only a small portion of public actions, and are also easy to solve. On the other hand, in **openstacks** all the information is public (agents are manager and manufacturer) resulting in little difference between the projected and distributed heuristic (except for the time necessary to evaluate a state).

In summary, the distributed heuristic is useful in domains with a high number of privately-dependent actions, which are a sign of necessary interaction among the agents, not visible to the projected heuristic, or with some crucial

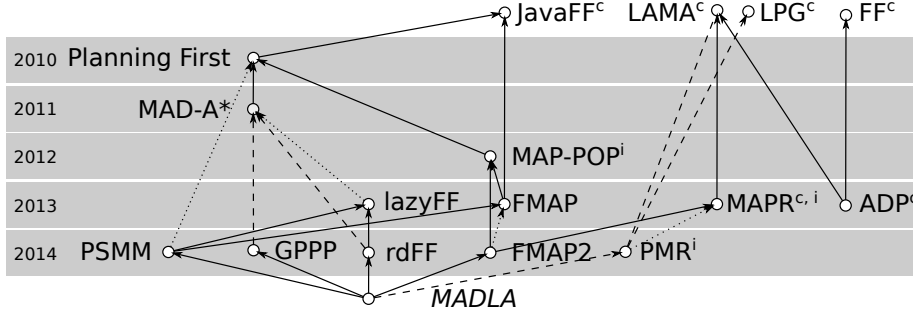


Figure 4: Subset of related work to the MADLA planner. The solid arrows represent comparison and improvement (in most cases); dashed lines represent informal comparison not over coverage (number of solved problems) or runtime; and dotted lines represent inspiration of the approaches. Additionally, superscripts *c*, *i* and *\** stand for centralized, incomplete and optimal algorithms respectively.

information being private (as in *woodworking*). In contrast, the projected heuristic performs better on domains where the agents are interchangeable (*driverlog*, *zenotravel*), or the distributed heuristic misguides the search (*depot*). In most cases, the MADLA search is able to let the better heuristic dominate the search and thus on most domains, MADLA dominates the single heuristic approaches. Notice, that on some domains MADLA even improves the coverage over the distributed heuristic (namely *blocksworld* and *elevators*). Only in significantly hard problems, where the distributed heuristic takes long to compute and does not give better estimates (*depot*), or the projected heuristic takes long to compute and misguides the search (*woodworking*), the MADLA search does not dominate the GBFS. It is also important to note that in *woodworking*, both projected and distributed FF heuristics ignore significant number of dead-ends (as described in Section 3.1.3), thus slowing the search and solving less problems.

#### 4.3. Comparison with the State of the Art

The planners for MA-STRIPS and related multiagent models in the literature spread from centralized single-core planners [18, 14], over parallel multi-core ones [3, 19], to fully distributed implementations with communication within one physical computer or over the network [3, 5, 12]. In Figure 4, we summarize relevant related planners to our MADLA planner (i.e., compatible with the MA-STRIPS model) with visualization of what planners were compared (and thus usually outperformed) which planners.

It is not surprising that the first MA-STRIPS planner called Planning First [20] was based on the principles used in the MA-STRIPS paper [2]. Planning First uses a coordination of local forward-search planners by a distributed solver of Constraint Satisfaction Problem (DisCSP). Planning First was a first representative *coordination-centric* planners and outperformed Java implementation of the FF planner JavaFF [21]. Besides coordination centric Distributed

Planning by Graph Merging (DPGM) [8, 22], Best Response Planning [23], and  $\mu$ -SATPLAN [24] a planner Distoplan [25] and following A# planner [26] pioneered an idea of optimal planning by intersection of Finite State Machines (FSM) representing the local plans of the agents. This idea was extended and practically refined in a satisficing planner based on nondeterministic FSMs representing local agents’ plans and its merging—Planning State Machine Merging (PSMM) in [19].

The multiagent state-space search was firstly used in distribution of optimal A\* search called MAD-A\* [3] with admissible projected landmark heuristics LM-cut [27] and Merge&Shrink [28]. Multiagent state-space search was also adopted by specific set of multiagent planners running as centralized sequential processes, therefore these planners are not directly comparable with the distributed multiagent planners as they have no communication overheads and runs on one thread. These planners are highly efficient algorithms able to outperform the state-of-the-art centralized planners as LAMA [16], FF [6], or LPG [29]. Multi-Agent Planning by Plan Reuse (MAPR) [14] is an incomplete approach, based on best-response principle, sequentially using a plan repairing algorithm (particularly LPG-adapt [30]) and its variation called Plan Merging by Reuse (PMR) [31] proposes a distribution scheme for MAPR. Agent decomposition-based planner (ADP) [18] uses Relaxed Planning Graphs for repeated extraction of subgoals of a multiagent planning problem and the FF planner to sequentially solve the extracted goals. A recent distributed multiagent state-space search planner inspired by MAD-A\* is called Greedy Privacy Preserving Planner (GPPP) [15], it uses landmarks refined for particular agents and is based on an iterative deepening backtrack search in relaxed planning subproblems. These planners are representative of *agent-centric* planners as the search in each agent drives the process of planning. The proposed MADLA planner fits this category.

The multiagent partial-order planners were firstly studied as a multiagent temporal planner TFPOP [32], which required extended model in contrast to MA-STRIPS because of additional requirement on temporal constraints. A series of partial-order planners based on model compatible with MA-STRIPS begun with incomplete MAP-POP [33] and two versions of practically improved planners FMAP [34, 12]. FMAP uses a parallelized local forward search to successively complete partial plans of the agents. The coordination subproblem is driven by a distributed form of heuristic utilizing Domain Transition Graphs (DTGs) for approximation of relaxed plans [9]. The computation of the heuristic can be parallelized as well. FMAP can be understood as both coordination centric and agent centric as the planning process follows the former and the heuristics the latter paradigm.

In Table 3, we show comparison of problems solved by MADLA, and four complete and distributed multiagent planners. The results show that MADLA looses considerably in `woodworking08` and `openstacks` against all planners supporting action costs. As mentioned in the previous section, these domains contain substantial number of dead-ends of which the FF heuristic (especially in the projected form) is oblivious.

Although the result table does not contain the PMR planner, MADLA out-

domain	rdFF	GPPP	PSMM	FMAP	MADLA
blocksworld (35)	6.8	3	25	19	<b>34.2</b>
depot (20)	6.2	8	0	6	<b>12.4</b>
driverlog (20)	14	9	13	15	<b>15.8</b>
elevators08 (30)	2.9	16 <sup>†</sup>	4	<b>30</b>	29.7
logistics00 (20)	5.8	<b>20</b>	9	10	<b>20</b>
openstacks (30)	11.7	0 <sup>‡</sup>	<b>30</b>	23	17.5
rovers (20)	14.7	10	14	19	<b>20</b>
satellites (20)	10.8	16	8	16	<b>20</b>
woodworking08 (30)	5.6	0 <sup>‡</sup>	<b>25</b>	22	4.5
zenotravel (20)	6.1	<b>20</b>	17	18	19
total (245)	84.6	102	145	178	<b>192.9</b>

Table 3: Comparison MADLA and state-of-the-art planners. <sup>†</sup>Used version of the domain in GPPP experiments without action costs, consisting of 16 problems. <sup>‡</sup>GPPP does not support action costs.

performs it on the presented benchmark set as well, just by the fact that PMR is an incomplete planner as stated in [14, 31]. PMR solves only problems where each goal fact is solvable by a single agent. Thus it does not solve problems of depot, logistics00, openstacks, and woodworking08 domains. Even if PMR solved all problems of all other domains, MADLA would outperform it by 35%.

Against rdFF [5], our recent multiagent heuristic search using different distribution scheme and implementation of FF, MADLA shows more than 2× improvement consistently over all domains with exception of woodworking08. Similarly, MADLA outperforms GPPP nearly 2× over all domains and PSMM by 33%. Finally, MADLA solves nearly 15 more problems of the benchmark set in contrast to currently top performing multiagent planner FMAP, which correspond to 8% improvement.

## 5. Conclusion

Similarly as for classical planning, heuristic state-space search is a viable technique for multiagent planning as well. However, in contrast to the classical heuristic search, the multiagent setup raises its own challenges. The dilemma of highly informed but slow versus less informed but fast heuristic estimators is manifested in the dichotomy of projected heuristic restricted to the agent’s local view of the problem versus distributed heuristic estimating the global heuristic value at the cost of significant communicational or computational overhead. The MADLA planner combines both fast local projection of the FF heuristic with a global distributed FF heuristic in an attempt to combine their benefits and mitigate their negative effects.

The technique used in MADLA to combine the heuristic estimators is based on the classical multi-heuristic search, but it does not evaluate all states by both heuristics. Instead, the local heuristic is used only while the distributed



heuristic is idle, that is waiting for replies from other agents. The projected heuristic is used to fully utilize the computational resources of the agent, even for less-informed, but faster search of the state space of the individual agent. When the estimation of the more-informed heuristics is finished, the evaluated state is used in both searches. This principle was theoretically analyzed in a general state-space search framework and practically evaluated in form of a MA-STRIPS based planner, which outperforms all current multiagent planners in a coverage metric over a common benchmark set.

Two research directions are left for future work. First, whether the principle of MADLA search can be used for optimal multiagent planning with similarly promising results. Second, as the results show, the planner does not perform well on domains with dead-ends, which relaxation heuristics are oblivious to. We can ask what efficiency boost could be achieved by combination not only of one heuristic as projected and distributed estimators, but also with other heuristics possibly orthogonal to the first heuristic pair; landmarks are an obvious choice.

Generally, utilizing the principle of combining fast and less accurate and slow but more informed heuristics in an asynchronous manner may be also an interesting research direction in the classical planning (esp. on multicore machines) and search in general.

## Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 13-22125S), by the Air Force Office of Scientific Research, USAF (grant no. FA8655-12-1-2096), and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

## References

## References

- [1] R. Fikes, N. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, in: Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71), 1971, pp. 608–620.
- [2] R. I. Brafman, C. Domshlak, From one to many: Planning for loosely coupled multi-agent systems, in: Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08), 2008, pp. 28–35.
- [3] R. Nissim, R. I. Brafman, Multi-agent A\* for parallel and distributed systems, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12), 2012, pp. 1265–1266.

- [4] R. Nissim, R. Brafman, Distributed heuristic forward search for multi-agent planning, *JAIR* 51 (2014) 293–332.
- [5] M. Štolba, A. Komenda, Relaxation heuristics for multiagent planning, in: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS’14)*, 2014, pp. 298–306.
- [6] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, *JAIR* 14 (2001) 253–302. doi:10.1613/jair.855.
- [7] M. Štolba, A. Komenda, Fast-forward heuristic for multiagent planning, in: *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP’13)*, 2013, pp. 75–83.
- [8] D. Pellier, Distributed planning through graph merging, in: *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence (ICAART 2010)*, 2010, pp. 128–134. doi:10.5220/0002702601280134.
- [9] M. Helmert, The Fast Downward planning system, *JAIR* 26 (2006) 191–246.
- [10] G. Röger, M. Helmert, The more, the merrier: Combining heuristic estimators for satisficing planning, in: *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS’10)*, 2010, pp. 246–249.
- [11] T. Bylander, The computational complexity of propositional STRIPS planning, *AIJ* 69 (1-2) (1994) 165–204.
- [12] A. Torreño, E. Onaindia, O. Sapena, FMAP: Distributed cooperative multi-agent planning, *Applied Intelligence* 41 (2) (2014) 606–626. doi:10.1007/s10489-014-0540-2.
- [13] E. Keyder, H. Geffner, Heuristics for planning with action costs revisited., in: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI’08)*, 2008, pp. 588–592.
- [14] D. Borrajo, Plan sharing for multi-agent planning, in: *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP’13)*, 2013, pp. 57–65.
- [15] S. Maliah, G. Shani, R. Stern, Privacy preserving landmark detection, in: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI’14)*, 2014, pp. 597–602.
- [16] S. Richter, M. Westphal, The LAMA planner: Guiding cost-based anytime planning with landmarks, *JAIR* 39 (1) (2010) 127–177.
- [17] International planning competition (2014).  
URL <http://ipc.icaps-conference.org/>

- [18] M. Crosby, M. Rovatsos, R. Petrick, Automated agent decomposition for classical planning, in: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13), 2013, pp. 46–54.
- [19] J. Tozicka, J. Jakubuv, A. Komenda, Generating multi-agent plans by distributed intersection of Finite State Machines, in: Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), 2014, pp. 1111–1112. doi:10.3233/978-1-61499-419-0-1111.
- [20] R. Nissim, R. I. Brafman, C. Domshlak, A general, fully distributed multi-agent planning algorithm, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), 2010, pp. 1323–1330.
- [21] A. Coles, M. Fox, D. Long, A. Smith, Teaching forward-chaining planning with javaff, in: Colloquium on AI Education, Twenty-Third AAAI Conference on Artificial Intelligence, 2008.
- [22] K. Durkota, A. Komenda, Deterministic multiagent planning techniques: Experimental comparison (short paper), in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 43–47.
- [23] A. Jonsson, M. Rovatsos, Scaling up multiagent planning: A best-response approach, in: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11), 2011, pp. 114–121.
- [24] Y. Dimopoulos, M. A. Hashmi, P. Moraitis, Extending SATPLAN to multiple agents, in: Proceedings of the 30th International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI'10), 2010, pp. 137–150.
- [25] E. Fabre, L. Jezequel, P. Haslum, S. Thiébaux, Cost-optimal factored planning: Promises and pitfalls, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10), 2010, pp. 65–72.
- [26] L. Jezequel, E. Fabre, A#: A distributed version of A\* for factored planning, in: Proceedings of the 51th IEEE Conference on Decision and Control, (CDC'12), 2012, pp. 7377–7382.
- [27] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: What's the difference anyway?, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09), 2009, pp. 162–169.
- [28] M. Helmert, P. Haslum, J. Hoffmann, Flexible abstraction heuristics for optimal sequential planning, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07), 2007, pp. 176–183.

- [29] A. Gerevini, I. Serina, LPG: A planner based on local search for planning graphs with action costs, in: Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS'02), 2002, pp. 13–22.
- [30] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06), 2006, pp. 212–221.
- [31] N. Luis, D. Borrajo, Plan merging by reuse for multi-agent planning, in: Proceedings of the 2nd ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'14), 2014, pp. 38–44.
- [32] J. Kvarnström, Planning for loosely coupled agents using partial order forward-chaining, in: Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS'11), 2011, pp. 138–145.
- [33] A. Torreño, E. Onaindia, O. Sapena, An approach to multi-agent planning with incomplete information, in: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12), 2012, pp. 762–767. doi:10.3233/978-1-61499-098-7-762.
- [34] A. Torreño, E. Onaindia, O. Sapena, FMAP: A heuristic approach to cooperative multi-agent planning, in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 84–92.

# Fault Tolerant Planning: Complexity and Compilation

Carmel Domshlak

Technion

Haifa, Israel

dcarmel@ie.technion.ac.il

## Abstract

In the context of modeling and reasoning about agent actions, contingent and classical planning can often be respectively seen as adopting “extreme pessimism” and “extreme optimism” about the action outcomes. For many everyday scenarios of human reasoning (and thus for many types of autonomous systems), both these approaches are just too extreme. Following Jensen, Veloso, and Bryant (2004), we examine a planning model that interpolates between classical and contingent planning via tolerance to arbitrary  $\kappa$  faults occurring during plan execution. We show that an important fragment of this *fault tolerant planning (FT-planning)* exhibits both an appealing solution structure, as well as appealing worst-case time-complexity properties. We also show that such FT-planning tasks can be efficiently compiled into classical planning as long as the number of possible faults per operator is bounded by a constant, and we show that this compilation can be attractive in practice.

## Introduction

To date, contingent and classical planning appear to be the two major approaches to non-probabilistic planning under full observability. In contingent planning, at least some aspects of system dynamics are modeled by operators with non-deterministic effects, and a plan should guarantee reaching a goal state under any realization of the actions it prescribed. In classical planning, the operators are all set to be deterministic, modeling only the singular intended effects of each action. While contingent plans provide much stronger guarantees on reaching the goal with respect to the true physics of the modeled system, they are also much harder to generate (both worst-case and empirically), and quite often they may simply not exist.

In the physical world, no actions are really guaranteed to succeed. However, non-determinism in real-world domains is often caused by infrequent errors that make otherwise deterministic operators fail. Hence, many unsolvable contingent planning tasks become solvable if we *assume* that *no more than some  $\kappa$  exceptional/faulty action effects will occur along the purported plan to the goal*. In the past, this observation brought numerous researchers to consider explicit representation and reasoning about faults of agents’

actions (Georgeff and Lansky 1986; Williams et al. 2003; Giunchiglia, Spalazzi, and Traverso 1994). In particular, Jensen, Veloso, and Bryant (2004) suggested a model of *fault tolerant planning (FT-planning)*, and developed first algorithms for generating plans that are robust for a single fault occurring during plan execution. This model is of our focus here.

Departing from contingent planning and generalizing the FT-planning model of Jensen, Veloso, and Bryant (2004), we show that, while FT-planning remains in general as computationally hard as contingent planning, one of its practically most valuable fragments, namely the one considered by Jensen, Veloso, and Bryant (2004), (1) is in PSPACE, (2) falls into NP when restricted to plans with only polynomial-length executions, and (3) is guaranteed to admit stationary solutions for solvable problems, solutions that sometimes induce (possibly cyclic) strong contingent plans. Furthermore, we show that these FT-planning tasks can be *efficiently* compiled into equivalent classical planning tasks in a way that is *sound, complete*, and *practicable*.

Our results join a growing body of work on planning under uncertainty and/or partial observability via compilation to classical planning (Palacios and Geffner 2009; Albore, Palacios, and Geffner 2009; Bonet and Geffner 2011; Brafman and Shani 2012a; Taig and Brafman 2013). At a high level, FT-planning is an instance of “assumption-based planning,” and the latter term has already been used for a broad range of ideas and techniques (Albore and Bertoli 2004; 2006; Albore and Geffner 2009; Bonet and Geffner 2011; Göbelbecker, Gretton, and Dearden 2011; Davis-Mendelow, Baier, and McIlraith 2012). Closest in spirit to our work here—though in two different ways—are probably the works of Albore and Bertoli (2006) and Davis-Mendelow, Baier, and McIlraith (2012). Albore and Bertoli suggested an interesting planning approach in which assumptions about operator effects are provided a priori as a linear temporal logic formula, and the planner takes these assumptions as axioms. In the worst-case, however, this approach remains as hard as contingent planning. Davis-Mendelow et al. exploit assumption-based assertions about the initial state to suggest a middle-ground between classical planning and conformant, or zero observability, planning. The latter, however, is very different from contingent planning, both conceptually and complexity-wise (Bonet 2010).

## Planning Formalisms and Solution Concepts

Non-deterministic planning tasks with full observability correspond to succinctly represented, goal-oriented non-deterministic Markov decision processes (Puterman 1994). Several languages for succinctly representing such tasks are in use (Hoffmann and Brafman 2005; Bonet 2010; Davis-Mendelow, Baier, and McIlraith 2012). To simplify presentation, here we adopt a minimalistic extension of STRIPS (Fikes and Nilsson 1971) to non-deterministic operator effects.

A **planning task** is given by a quadruple  $\Pi = \langle P, O, s_0, G \rangle$ .  $P$  is a set of  $n$  propositions, with world states  $S$  being represented by complete valuations of  $P$ , and usually discussed as sets of propositions that hold true in them.  $s_0 \in S$  is an initial state<sup>1</sup>, and  $G$  is a subset of  $P$ : a state  $s$  is a goal state iff  $G \subseteq s$ , and the set of all goal states is denoted by  $S_G$ .  $O$  is a set of operators  $o = \langle \text{pre}(o), \text{eff}(o) \rangle$  where the *precondition*  $\text{pre}$  is a subset of propositions  $P$ , and  $\text{eff} = \{e_1, \dots, e_{n_o}\}$  is a set of *possible* effects of  $o$ . Each possible effect  $e \in \text{eff}$  is given by a pair  $\langle \text{add}(e), \text{del}(e) \rangle$  of subsets of  $P$ , corresponding to its add and delete lists, respectively. An operator  $o$  is applicable in state  $s$  iff  $\text{pre}(o) \subseteq s$ , and the set of all such operators is denoted by  $O(s)$ . If  $o \in O(s)$  is applied in  $s$ , it changes the world to *one of* the states  $\text{Res}[s; o] = \bigcup_{e \in \text{eff}(o)} \{\text{Res}[s; e]\}$ , where  $\text{Res}[s; e] = (s \setminus \text{del}(e)) \cup \text{add}(e)$  is the state resulting from the effect  $e$  occurring in  $s$ . A (contingent) **plan** for a task  $\Pi$  is an action strategy that guarantees reaching a goal state  $s \in S_G$  from  $s_0$ , under any realization of the operators applied along the way; the process of search for contingent plans is called **contingent planning**.

While nondeterministic operators must be dealt with in one form or another in many planning applications, two problems particular to contingent planning must be taken into account. First, deciding whether a contingent plan exists is EXPTIME-complete (Rintanen 2004).<sup>2</sup> Second, many tasks admit no contingent plans, and this is true even for simple tasks that humans feel comfortable dealing with (Cimatti et al. 2003; Pistore and Vardi 2007). In that respect, a pragmatic alternative to contingent planning is **classical planning**, operators are deterministic. By adopting classical planning as an abstraction of contingent planning, we *assume* that we know precisely what will happen when an operator  $o$  is applied in state  $s$ . This assumption is then “encoded” at the level of individual operators by what is called *determinization*, reducing the set of possible effects of each operator to exactly one effect.

While being much more restricted, classical planning resolves to a large extent the two aforementioned shortcomings of contingent planning. First, restricting each state/action pair to a sole possible successor often renders unsolvable problems solvable. Second, classical planning is in PSPACE (Bylander 1994), and more importantly, it is

in NP if restricted to polynomial-length plans. Last but not least, classical plans are structurally simple, constituting linear sequences of operators. Together with NP-membership, this structural simplicity allows for exploiting various OR-graph search techniques for developing empirically efficient solvers for classical planning (Hoffmann and Nebel 2001; Helmert 2006; Rintanen, Heljanko, and Niemelä 2006; Kissmann and Edelkamp 2012). As a result, combining classical planning with online re-planning in unexpected situations is a popular and effective approach to closed-loop control of autonomous systems (Yoon et al. 2008; Talamadupula et al. 2010; Domshlak et al. 2011; Bonet and Geffner 2011; Brafman and Shani 2012b).

## Fault Tolerant (Contingent) Planning

Given the relative pros and cons of contingent and classical planning, the first question one might ask is: If these are the two extremes, how can we interpolate between them in a simple and useful manner? This question brought us to consider *fault tolerant planning*: planning under the assumption that no more than some  $\kappa$  unintended effects of the operators will occur along the purported plan to the goal, but at the same time, under a requirement for the plans to be provably robust for up to  $\kappa$  such operator faults during plan execution. Fault tolerant planning was originally introduced by Jensen, Veloso, and Bryant (2004) in order to bring some key information from probabilistic uncertainty models to qualitative non-deterministic planning. The basic idea is to associate the contingent planning task at hand with an explicit distinction between the primary and exceptional effects of its operators. The model we adopt for that purpose is simply a function  $\mathcal{F}$  that maps each possible effect of each operator to the “number of exceptions,” or unintended artifacts, associated with this effect. The operator effects  $e$  for which  $\mathcal{F}(e) = 0$  correspond to the primary effects of the respective operators in  $s$ .

**Definition 1** Let  $\Pi = \langle P, O, s_0, G \rangle$  be a contingent planning task. An **exception model** for  $\Pi$  is a function  $\mathcal{F} : \bigcup_{o \in O} \text{eff}(o) \rightarrow \mathbb{N}$ , computable in time polynomial in  $|\Pi|$ . If, for each operator  $o \in O$ ,  $|\{e \mid e \in \text{eff}(o), \mathcal{F}(e) = 0\}| \leq \alpha$ , then  $\mathcal{F}$  is called  **$\alpha$ -primary**. Likewise, if, for each operator  $o \in O$ ,  $|\{e \mid e \in \text{eff}(o), \mathcal{F}(e) = 0\}| > 0$ , then  $\mathcal{F}$  is called **normative**.

In simple terms, an exception model is  $\alpha$ -primary if at most  $\alpha$  effects of each operator are considered to be its primary effects, and it is normative if each operator is associated with at least one primary effect. In these terms, the work of Jensen, Veloso, and Bryant (2004) has been devoted to fault tolerant planning under 1-primary normative exception models, which seem to cover well operator non-determinism that stems from physical complications of executing agent actions in the real world.<sup>3</sup> Associating non-deterministic operators with exception models allows for a

<sup>1</sup>We assume here that there is no uncertainty about the initial state, and later discuss the impact of this assumption.

<sup>2</sup>EXPTIME-completeness still holds even for testing the existence of plans that reach the goal with probability exceeding  $p$  for probabilistic problems with full observability (Littman 1997).

<sup>3</sup>If, however, some operators model knowledge acquisition, i.e., sensing, then (part of the) operator non-determinism will be due to primary operator effects, and thus planning with  $\alpha$ -primary models for  $\alpha > 1$  is not of theoretical interest only.

simple relaxation of contingent planning to planning under fault tolerance requirements as above. Let  $\Pi$  be a contingent planning task,  $\mathcal{F}$  be an exception model for  $\Pi$ , and  $\pi$  be an action policy for  $\Pi$ . Overloading the notation, for an execution  $\rho = (s_0, e_0, \dots, s_i, e_i, \dots)$  of  $\pi$ , we define  $\mathcal{F}(\rho) \triangleq \sum_{i=0}^{\infty} \mathcal{F}(e_i)$ .

- An execution  $\rho$  of  $\pi$  is called  $\kappa$ -**admissible** if  $\mathcal{F}(\rho) \leq \kappa$ .
- Action policy  $\pi$  is a  $\kappa$ -**plan** for  $\Pi$  if each of its  $\kappa$ -admissible executions is finite and reaches the goal.

In what follows, we refer to triplets  $\langle \Pi, \mathcal{F}, \kappa \rangle$  as above as **fault tolerant (FT) planning tasks**, and solutions for such tasks are precisely  $\kappa$ -plans for  $\Pi$  under  $\mathcal{F}$ .

In general,  $\kappa$ -plan  $\pi$  can be either a stationary (possibly partial) policy  $\pi : S \rightarrow O$ , or a non-stationary policy  $\pi : S \times \mathbb{N} \rightarrow O$  that depends on the current state and the number of “failures so far.” As noted by Jensen et al. (2004), the latter can be captured as a stationary policy for a certain contingent planning task  $\Pi^{(\mathcal{F}, \kappa)}$  that we refer to as  $(\mathcal{F}, \kappa)$ -**reformulation** of  $\Pi$ : Given a FT-planning task  $\langle \Pi = \langle P, O, s_0, G \rangle, \mathcal{F}, \kappa \rangle$ ,  $\Pi^{(\mathcal{F}, \kappa)}$  is a contingent planning task over states  $S^{(\mathcal{F}, \kappa)} = S \times \{0, \dots, \kappa\}$ , operators  $O$ , initial state  $s_0^{(\mathcal{F}, \kappa)} = (s_0, 0)$ , and goal states  $S_G^{(\mathcal{F}, \kappa)} = \bigcup_{i=0}^{\kappa} \{(s, i) \mid s \in S_G\}$ . For each  $s \in S$ ,  $o \in O(s)$ , and  $0 \leq i \leq \kappa$ ,  $o$  is applicable in  $(s, i)$ , and if applied, it changes the world to one of the states

$$Res[(s, i); o] = \bigcup_{\substack{e \in \text{eff}(o), \\ \kappa - (i + \mathcal{F}(e)) \geq 0}} \{Res[(s, i); e]\}, \quad (1)$$

where  $Res[(s, i); e] = (Res[s; e], i + \mathcal{F}(e))$ .

It is not hard to verify that there is a bijective correspondence between plans for  $\Pi^{(\mathcal{F}, \kappa)}$  and *non-stationary*  $\kappa$ -plans for  $\langle \Pi, \mathcal{F}, \kappa \rangle$ , but the relation to stationary  $\kappa$ -plans for  $\langle \Pi, \mathcal{F}, \kappa \rangle$  is less immediate. Theorem 1 clarifies this matter. Let  $\pi$  be a (not necessarily a plan) policy for  $\Pi^{(\mathcal{F}, \kappa)}$ . The *execution tree*  $T_\pi(s, i)$  is the tree of possible executions of  $\pi$  starting at  $(s, i)$ , with nodes corresponding to states of  $\Pi^{(\mathcal{F}, \kappa)}$ , edges corresponding to operator effects, and  $S_\pi^{(\mathcal{F}, \kappa)} \subseteq S^{(\mathcal{F}, \kappa)}$  denoting the set of *internal* nodes of  $T_\pi$ . (Henceforth,  $T_\pi(s_0, 0)$  is referred to for short as  $T_\pi$ .)

**Theorem 1** *Let  $\langle \Pi, \mathcal{F}, \kappa \rangle$  be a solvable FT-planning task. If  $\mathcal{F}$  is normative, then there exists a stationary  $\kappa$ -plan  $\pi$  for  $\langle \Pi, \mathcal{F}, \kappa \rangle$ . In contrast, there exist solvable FT-planning tasks (with nonnormative exception models) for which there are no stationary  $\kappa$ -plans.*

The proof of the second sub-claim is by example: Let  $\Pi$  be a contingent planning task over states  $S = \{s_0, \dots, s_6\}$ , operators  $O = \{o_0, \dots, o_5\}$  as in Figure 1a, initial state  $s_0$ , and  $S_G = \{s_6\}$ . Assume an exception model  $\mathcal{F}$  for  $\Pi$  as in the last column of Figure 1a. Figure 1b depicts the only contingent plan for the reformulation  $\Pi^{(\mathcal{F}, 1)}$ , and the respective 1-plan for  $\langle \Pi, \mathcal{F}, 1 \rangle$  is not stationary: different actions,  $o_1$  and  $o_2$ , are taken at state  $s_1$  with 0 and 1 “exceptions so far,” respectively, and such a history-dependent choice of operator at  $s_1$  is unavoidable.

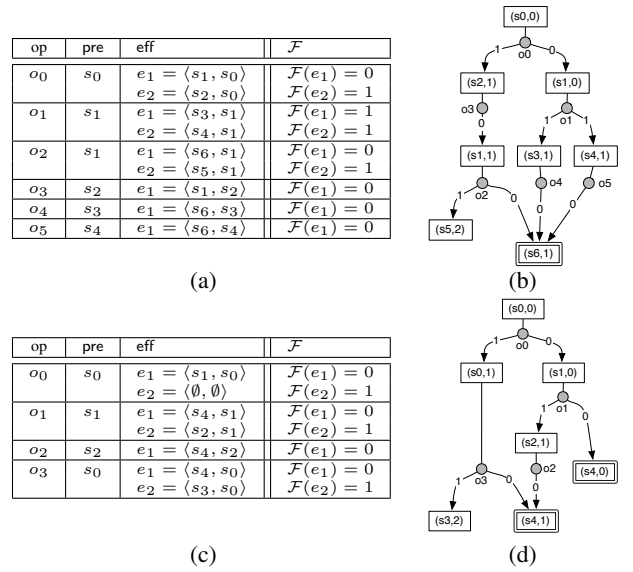


Figure 1: Illustrations for the examples around Theorem 1.

Note that  $\mathcal{F}$  in the above example is not normative because neither of the effects of operator  $o_1$  is primary. With normative exception models, the situation is indeed different. Let  $\langle \Pi, \mathcal{F}, \kappa \rangle$  be a solvable FT-planning task with a normative model  $\mathcal{F}$ , and  $\pi$  be a contingent plan for  $\Pi^{(\mathcal{F}, \kappa)}$ . If  $\pi(s, i) = \pi(s, j)$  for all pairs of reformulation states  $(s, i), (s, j) \in S_\pi^{(\mathcal{F}, \kappa)}$ , then we are done because the  $\kappa$ -plan for  $\langle \Pi, \mathcal{F}, \kappa \rangle$  corresponding to  $\pi$  is stationary. Otherwise, let  $(s, i), (s, j) \in S_\pi^{(\mathcal{F}, \kappa)}$ ,  $i < j$  be a pair of reformulation states for which  $\pi(s, i) \neq \pi(s, j)$ . The proof is accomplished by showing that  $\pi'$ , obtained from  $\pi$  by replacing  $\pi(s, j)$  with  $\pi(s, i)$ , is also a plan for  $\Pi^{(\mathcal{F}, \kappa)}$ , and thus we can always iteratively reduce a non-stationary  $\kappa$  to a stationary one.

Note that, by the construction used in the proof of Theorem 1, if  $\pi$  is a non-stationary  $\kappa$ -plan for a FT-planning task  $\langle \Pi, \mathcal{F}, \kappa \rangle$  with a normative model  $\mathcal{F}$ , then  $\pi$  can be efficiently translated into a stationary  $\kappa$ -plan  $\pi'$  for  $\langle \Pi, \mathcal{F}, \kappa \rangle$ . Likewise, for some of such pairs  $\pi$  and  $\pi'$ ,  $\pi'$  may turn out to be a strong cyclic contingent plan for  $\Pi$ . For instance, let  $\Pi$  be a contingent planning task over states  $S = \{s_0, \dots, s_4\}$ , operators  $O = \{o_0, \dots, o_3\}$ , initial state  $s_0$ , and  $G = s_4$ . The operators are defined as in the table in Figure 1c, and the exception model  $\mathcal{F}$  associated with  $\Pi$  is given in the last column of that table. Figure 1d depicts a contingent plan  $\pi$  for the reformulation  $\Pi^{(\mathcal{F}, \kappa)}$ , and the respective 1-plan for  $\langle \Pi, \mathcal{F}, 1 \rangle$  is not stationary: different actions,  $o_0$  and  $o_3$ , are taken at state  $s_0$  with 0 and 1 “exceptions so far,” respectively. However, if we modify  $\pi$  as in the proof of Theorem 1, the resulting plan  $\pi'$  for  $\Pi^{(\mathcal{F}, \kappa)}$  will induce a strong cyclic contingent plan  $\pi^*(s_i) = o_i$  for  $\Pi$ .

## Complexity and Compilation

Two decision problems are of interest in the context of FT-planning: Let  $\Pi$  be a contingent planning task, and  $\mathcal{F}$  be an  $\alpha$ -primary model for  $\Pi$ .

**FT-PLAN- $\alpha$ - $\kappa$ :** Does  $\Pi$  have a  $\kappa$ -plan?

**POLY-FT-PLAN- $\alpha$ - $\kappa$ :** Does  $\Pi$  have a  $\kappa$ -plan such that all its  $\kappa$ -admissible executions reach the goal after a polynomial number of steps?

At first view, the effective difference between FT-PLAN- $\alpha$ - $\kappa$  and contingent planning is not clear. In general, for sufficiently large values of  $\alpha$  (e.g.,  $\alpha = |S|$ ), contingent planning can trivially be reduced to FT-PLAN- $\alpha$ - $\kappa$  for *any*  $\kappa$ , and hence the latter decision problem is EXPTIME-hard. In fact, FT-PLAN- $\alpha$ - $\kappa$  can be polynomially reduced to FT-PLAN-2- $\kappa$  by simulating each operator with  $\alpha$  primary effects by a “ladder” of  $\log \alpha$  operators, each with at most two primary effects. Hence, even FT-PLAN-2- $\kappa$  is EXPTIME-hard. However, while the definition of exception models is rather general, the specific settings that brought us to this investigation correspond to the normative 1-primary exception models considered by Jensen, Veloso, and Bryant (2004). In what comes next, we focus on that fragment of FT-planning.<sup>4</sup>

### FT-planning with 1-primary models

Unlike FT-PLAN-2- $\kappa$ , FT-PLAN-1- $\kappa$  nicely generalizes classical planning, which simply corresponds to first associating the contingent planning task with a normative 1-primary model, and then adopting the extreme optimism that no failures will occur along the purported plan to the goal. In other words, *the decision version of classical planning is precisely the FT-planning class FT-PLAN-1-0*. At the same time, FT-PLAN-1-1 already goes way beyond classical planning. While plans for FT-PLAN-1-1 are restricted to at most one operator failure per possible plan execution, these failures are bounded neither to specific operators nor to specific stages of the purported plan. Hence, while plans for FT-PLAN-1-0 are linear sequences of actions, plans for FT-PLAN-1- $\kappa$  with  $\kappa > 0$  are tree-structured, and may actually exhibit substantial branching: unlike in (EXPSPACE-complete) planning under the  $k$ -branching assumption (Bonet 2010), plans for FT-PLAN-1- $\kappa$  may have to always interleave between acting and branching, even for  $\kappa = 1$ .

For example, suppose that a robot should move from  $x_1$  to  $x_5$  on the map depicted in Figure 2(a). Movements on the segments  $(x_1, x_2)$  and  $(x_1, x_3)$  are considered safe and thus are modeled by deterministic operators. Movements  $move(x_i, x_j)$  on the other three segments are modeled by non-deterministic operators with three possible effects:  $move(x_i, x_j)$  typically brings the robot to  $x_j$ , with no side effects, but it may also bring the robot to  $x_j$  with a flat tire, or keep it at  $x_i$  for the same reason. Initially the robot has no flats, but also no spare tires. A single spare tire can be picked up at each of the two intermediate locations  $x_2$  and  $x_3$ . Figures 2(b-d) depict stationary 0-plan, 1-plan, and 2-plan for the respective FT-planning tasks, under

<sup>4</sup>While the motivation for the FT-PLAN-1- $\kappa$  fragment comes from its excellent applicability in practice, it is obviously not the only fragment of FT planning to be so motivated. For instance, if some operators model knowledge acquisition, i.e., sensing, then (part of the) operator non-determinism will be due to primary operator effects, and thus planning with  $\alpha$ -primary models for  $\alpha > 1$  is not of a theoretical interest only.

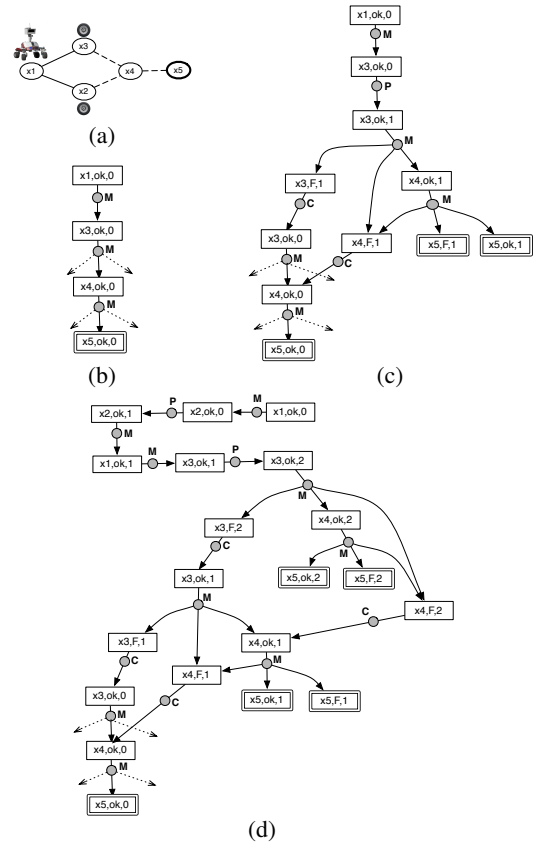


Figure 2:  $\kappa$ -plans for an inline example

a normative 1-primary exception model that maps the primary effects of all actions to 0, and the exceptional effects of the non-deterministic move actions to 1. In the triplet denotation  $[x, y, z]$  of the states,  $x$  is the robot’s location,  $y \in \{ok, F\}$  is the status of the tire, and  $z$  is the number of spare tires in the robot’s possession. State representation in this problem also addresses the availability of the spare tires at  $x_2$  and  $x_3$ , but we omit this information in the figure for brevity. The dashed arrows depict possible effects of the actions that the agent ignores at planning. The 0-plan is a classical plan, and it is as simple as the example itself. The 1-plan is already more involved: to guarantee reaching  $x_5$  under a possibility of a single fault, the robot picks up a spare tire at  $x_3$ . In turn, the 2-plan in Figure 2d prescribes that the robot first collect both spare tires at  $x_2$  and  $x_3$ , and only then start moving towards  $x_5$ , replacing flat tires on the way, if needed.

Still, despite the structural complexity of  $\kappa$ -plans, the EXPTIME-hardness proof for FT-PLAN-2- $\kappa$  does not carry over to FT-PLAN-1- $\kappa$ , and for a good reason: Theorem 2 below shows that FT-PLAN-1- $\kappa$  is in PSPACE, that is, worst-case not harder than classical planning. Moreover, Theorem 3 then shows even closer resemblance between these two formalisms, namely that POLY-FT-PLAN-1- $\kappa$  is in NP.

**Theorem 2** FT-PLAN-1- $\kappa$  is in PSPACE.

A non-deterministic algorithm (that can also be compiled to a Turing machine) for deciding whether there is a  $\kappa$ -plan



**Algorithm** BO-PLAN-1- $\kappa(\Pi, \mathcal{F})$

```

main
  PLAN-FT( $s_0, \kappa$ )
accept

procedure Plan-FT( $s, k$ )
   $steps \leftarrow 0$ 
  while  $steps < 2^n$ 
  do {
    if  $s \models G$  then return
    choose operator  $o$  s.t.  $s \models \text{pre}(o)$ 
    for  $e \in \text{eff}(o)$ 
    do {
      if  $\mathcal{F}(e) > k$ 
      then continue // under assumed  $\kappa$ ,  $e$  cannot happen here
      else if  $\mathcal{F}(e) > 0$ 
      then Plan-FT( $\text{Res}[s; e], k - \mathcal{F}(e)$ )
      else { // proceed with the primal effect of  $\langle \text{pre}, \text{eff} \rangle$  at  $s$ 
         $steps \leftarrow steps + 1$ 
         $s \leftarrow \text{Res}[s; e]$ 
      }
    }
  }
reject

```

Figure 3: PSPACE algorithm for deciding FT-PLAN-1- $\kappa$ .

for an FT-planning task  $\langle \Pi, \mathcal{F}, \kappa \rangle$  with 1-primary  $\mathcal{F}$  and  $|P| = n$  is depicted in Figure 3. The respective Turing machine is in PSPACE because, at any point, there are at most  $\kappa$  open calls to the *Plan-FT* procedure, each storing a single state in  $n$  bits and a single counter *steps* in  $n$  bits. Finally, since it is in PSPACE, FT-PLAN-1- $\kappa$  is also PSPACE-complete by the PSPACE-hardness of classical planning under the description language we use, and equivalence of the latter to FT-PLAN-1-0.

**Theorem 3** POLY-FT-PLAN-1- $\kappa$  is in NP.

The proof is by showing that, if  $\langle \Pi, \mathcal{F}, \kappa \rangle$  has a  $\kappa$ -plan  $\pi'$  such that all its  $\kappa$ -admissible executions reach the goal after  $O(n^c)$  steps, then there is a  $\kappa$ -plan  $\pi$  with the same property such that  $\|\pi\| = O(b^{(\kappa+1)}n^{c(\kappa+2)})$ , where  $b = \max_{o \in O} |\text{eff}(o)|$ . Since  $b = O(\|\Pi\|)$  and both  $c$  and  $\kappa$  are  $O(1)$ , the rest stems from the standard guess-and-verify argument of NP-membership.

First, since  $\kappa$ -plans guarantee goal reachability only along executions with  $\kappa$  or fewer exceptions, let  $\pi$  follow  $\pi'$  on states reachable by the  $\kappa$ -admissible executions of  $\pi'$ , and make a random operator choice everywhere else. Thus, only the  $\kappa$ -admissible executions of  $\pi$  should be represented. Second, as we upper-bound the description size of  $\pi$ , for simplicity we assume (i) extensive, tree-structured representation of  $\pi$ , and (b) that the range of  $\mathcal{F}$  is  $\{0, 1\}$ . For  $0 \leq i \leq \kappa$ , let  $f(\pi, i)$  be the number of  $i$ -admissible executions of  $\pi$  that are not  $(i-1)$ -admissible. Clearly,  $\sum_{i=0}^{\kappa} f(\pi, i)$  is the overall number of  $\kappa$ -admissible executions, and thus  $\|\pi\| = O(n^c \sum_{i=0}^{\kappa} f(\pi, i))$ . Since  $\mathcal{F}$  is 1-primary, there is at most one 0-admissible execution of  $\pi$  per possible initial state, and thus  $f(\pi, 0) = 1$ . Recursively, due to the same argument of  $\mathcal{F}$  being 1-primary, each of the  $O(n^c)$  operator instances along that single 0-admissible execution may branch into  $b$  1-admissible executions of  $\pi$ . However, these are the only possible sources of 1-admissible executions. Thus,  $f(\pi, 1) = O(f(\pi, 0) \cdot bn^c) = O(bn^c)$ , and in general,

$f(\pi, i) = O((bn^c)^i)$ . Hence,  $\|\pi\| = O(n^c \cdot (bn^c)^{\kappa+1}) = O(b^{(\kappa+1)}n^{c(\kappa+2)})$ .

Note that, while  $\kappa = O(1)$  should suffice for most interests in practice, the PSPACE-membership result of Theorem 2 holds for  $\kappa = O(\text{poly}(\|\Pi\|))$ . This is not so, however, with the NP-membership result of Theorem 3, which relies upon  $\kappa = O(1)$ .

## Compilation to Classical Planning

Theorems 2 and 3 put FT-planning under 1-primary exception models rather close to classical planning. On the one hand, that suggests that classical planning machinery can possibly be adapted to solve such FT-planning tasks. On the other hand, the non-linearity of  $\kappa$ -plans under 1-primary models seems to complicate applying classical planning algorithms to FT-planning.

We now show that FT-planning under 1-primary models can be efficiently compiled into classical planning, at least as long as the number of non-deterministic effects per operator is bounded by a constant. The compilation is to STRIPS with negative preconditions and conditional effects. In this formalism, a planning task is given by a quadruple  $\Pi = \langle P, O, s_0, G \rangle$ , with  $P$ ,  $s_0$ , and  $G$  being as in our formalism for contingent planning. Operators  $o \in O$  are pairs  $\langle \text{pre}(o), \text{eff}(o) \rangle$  where the precondition  $\text{pre}(o)$  is a subset of literals over  $P$ , and  $\text{eff}(o)$  is a set of conditional effects. A conditional effect  $e$  is a triplet  $\langle \text{con}(e), \text{add}(e), \text{del}(e) \rangle$  of condition, add, and delete lists, respectively, where  $\text{con}(e)$  is a subset of literals over  $P$ , while  $\text{add}(e)$  and  $\text{del}(e)$  are subsets of propositions. Operator  $o$  is applicable in state  $s$  iff  $s \models \text{pre}(o)$ . If  $o \in O(s)$  is applied in  $s$ , it deterministically changes the world to state  $\text{res}[s; o] = \bigcup_{e \in \text{eff}(o), s \models \text{con}(e)} (s \setminus \text{del}(e)) \cup \text{add}(e)$ .

We start with a compilation of a simple fragment of FT-PLAN-1- $\kappa$ , corresponding to FT-planning tasks  $\langle \Pi = \langle P, O, s_0, G \rangle, \mathcal{F}, \kappa \rangle$  such that (i) each operator of  $\Pi$  has at most two effects, and (ii)  $\mathcal{F}$  is a normative 1-primary exception model for  $\Pi$  such that, for each  $o \in O(s)$ , if  $\text{eff}(o) = \{e_0\}$ , then  $\mathcal{F}(e_0) = 0$ , and if  $\text{eff}(o) = \{e_0, e_1\}$ , then  $\mathcal{F}(e_0) = 0$  and  $\mathcal{F}(e_1) = 1$ . We begin with an example that illustrates the basic idea behind this compilation. Let  $\kappa = 2$ , and let Figure 4a depict an irreducible contingent plan  $\pi$  for  $\Pi^{(\mathcal{F}, \kappa)}$ : the arcs correspond to the operator effects and are labeled with the respective values of  $\mathcal{F}$ , and double-frame states are the goal states.

Note that, in Figure 4a, the states and operator instances in  $\pi$  are numbered consistently with a DFS traversal of the execution tree  $T_\pi$ . Therefore, the operator sequence  $o_1, \dots, o_7$  induces a sequence of policies  $\pi_0, \dots, \pi_7$  for  $\Pi^{(\mathcal{F}, \kappa)}$  such that  $\pi_0$  is an empty policy,  $\pi_7 = \pi$ , and each  $\pi_i$  extends  $\pi_{i-1}$  with mapping a single leaf of  $T_{\pi_{i-1}}$  to operator  $o_i$ . An important property of this sequence of policies  $\pi_0, \dots, \pi_7$  is that, for  $0 \leq j \leq 2$ , each  $\pi_i$  induces *at most one* execution  $\rho$  with  $\mathcal{F}(\rho) = j$  that does not achieve the goal within  $T_{\pi_i}$ . The latter is emphasized by the tabular representation of this sequence of policies in Figure 4b. The columns in the table capture certain *subsets*  $\sigma_0, \dots, \sigma_7$  of leaves of  $T_{\pi_0}, \dots, T_{\pi_7}$ , that is, of the end-states of  $\pi_0, \dots, \pi_7$ , respectively. For

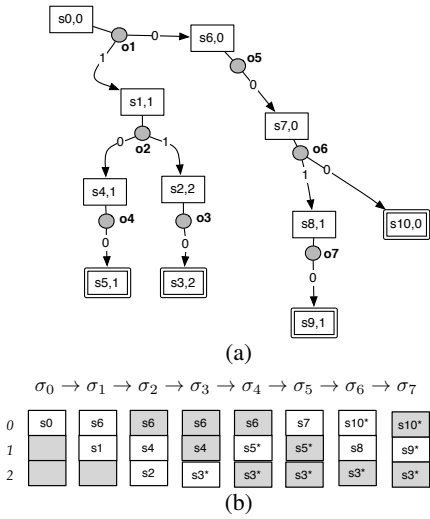


Figure 4: A plan  $\pi$  for  $\Pi^{(\mathcal{F}, \kappa)}$ , and the end-state set representation of the induced sequence of  $\pi$ 's sub-policies.

$0 \leq j \leq 2$ , each  $\sigma_i$  contains at most one state with “ $j$  failures so far,” denoted  $\sigma_i(j)$ , with  $\sigma_i(j) = \perp$  denoting that  $\sigma_i$  does not contain a state for  $j$ . For  $i = 0$ ,  $\sigma_0(0) = (s_0, 0)$ , and  $\sigma_0(1) = \sigma_0(2) = \perp$ . For  $i > 0$ , if  $T_{\pi_i}$  has a (unique) leaf  $(s, j)$  such that  $s \notin S_G$ , then  $\sigma_i(j) = (s, j)$ , and otherwise,  $\sigma_i(j) = \sigma_{i-1}(j)$ . Asterisks in the table are by the goal states, and note that the last set  $\sigma_7$  is the first in the sequence to contain only goal states.

It turns out that any irreducible  $\kappa$ -plan for any FT-planning task from the fragment in question induces such a DFS-ordered sequence of sub-policies with “at most one non-goal leaf with  $j$  failures so far.” It is precisely this property that provides a basis for our compilation of  $\langle \Pi = \langle P, O, s_0, G \rangle, \mathcal{F}, \kappa \rangle$  to an equivalent classical planning task  $\Pi' = \langle P', O', \sigma'_0, G' \rangle$ . For now, we postpone the formal statement and proof of this property, and formulate the actual compilation. After that, in Lemma 5, we formally state the task properties underlying the compilation, and use this lemma to specify compilation for a wider class of FT-planning tasks.

**Propositions.** For clarity, we use letters  $s$  and  $\sigma$  to denote states of  $\Pi$  and  $\Pi'$ , respectively. The subsets  $\sigma_0, \dots, \sigma_7$  of  $S^{(\mathcal{F}, \kappa)}$  in Figure 4b hint at the state space structure of  $\Pi'$ : Each reachable state  $\sigma$  of  $\Pi'$  represents a partial policy  $\pi_\sigma$  for  $\Pi^{(\mathcal{F}, \kappa)}$  that corresponds to a concrete stage of a certain DFS traversal of the purported tree-structured plan for  $\Pi^{(\mathcal{F}, \kappa)}$ . To support that,  $P'$  contains  $\kappa + 1$  replicas  $P_0, \dots, P_\kappa$  of  $P$ , as well as a set of auxiliary propositions  $\{open_i\}_{i=0}^\kappa$ . The interpretation of  $open_i \in \sigma$  is that “the policy for  $\Pi^{(\mathcal{F}, \kappa)}$  represented by  $\sigma$  induces an  $i$ -admissible execution that does not achieve the goal, and the end-state of that execution is captured by the values of  $P_i$  in  $\sigma$ .” In what follows, by  $\sigma/P_i$  we denote the valuation provided by  $\sigma$  to propositions  $P_i$ . Likewise, if  $\phi$  is a set of literals either over  $P$  or over one of the proposition sets  $P_0, \dots, P_\kappa$  of  $\Pi'$ , then, for  $0 \leq i \leq \kappa$ ,  $[\phi]_i$  is a set of literals over  $P_i$ , obtained from  $\phi$  by replacing all the propositional symbols with their

counterparts in  $P_i$ . For example,  $[\{p, \neg q\}]_i = \{p_i, \neg q_i\}$ , and  $[\{p_i, \neg q_i\}]_j = \{p_j, \neg q_j\}$ .

**Operators.**  $O'$  contains  $\kappa + 1$  sets of operators  $O_0, \dots, O_\kappa$ , with  $O_i = \{o_i \mid o \in O\}$ . For each  $o \in O$ , the precondition of the operator  $o_i$  is

$$\text{pre}(o_i) = [\text{pre}(o)]_i \cup \{open_i\} \cup \bigcup_{j=i+1}^\kappa \{\neg open_j\}.$$

In other words, if state  $\sigma$  of  $\Pi'$  represents a policy for  $\Pi^{(\mathcal{F}, \kappa)}$  such that some admissible executions of that policy do not reach the goal, then the planner is forced to extend the  $i$ -admissible such execution with the *highest* index  $i$ . If either  $o$  is deterministic or  $i = \kappa$ , then  $\text{eff}(o_i) = \text{Res}[\sigma/P_i; [e_0]_i]$ . Otherwise, if  $\text{eff}(o) = \{e_0, e_1\}$  and  $i < \kappa$ , then

$$\text{eff}(o_i) = \text{Res}[\sigma/P_i; [e_0]_i] \wedge [\text{Res}[\sigma/P_i; [e_1]_i]]_{i+1} \wedge open_{i+1}.$$

In the formalism of our choice, such operator effects are captured as follows. If  $\text{eff}(o) = \{e_0\}$  or  $i = \kappa$ , then  $\text{eff}(o_i)$  contains a single unconditional effect:

$$\text{eff}(o_i) = \{ \langle \emptyset, [\text{add}(e_0)]_i, [\text{del}(e_0)]_i \rangle \}.$$

Otherwise, if  $\text{eff}(o) = \{e_0, e_1\}$  and  $i < \kappa$ , then

$$\text{eff}(o_i) = \Phi_i \cup \left\{ \begin{array}{l} \langle \emptyset, [\text{add}(e_0)]_i, [\text{del}(e_0)]_i \rangle, \\ \langle \emptyset, \{open_{i+1}\}, \emptyset \rangle, \\ \langle \emptyset, [\text{add}(e_1)]_{i+1}, [\text{del}(e_1)]_{i+1} \rangle \end{array} \right\},$$

where

$$\Phi_i = \bigcup_{p \notin \text{add}(e_1) \cup \text{del}(e_1)} \left\{ \begin{array}{l} \langle \{p_i\}, \{p_{i+1}\}, \emptyset \rangle, \\ \langle \{\neg p_i\}, \emptyset, \{p_{i+1}\} \rangle \end{array} \right\}$$

compactly encodes the situation calculus frame axioms between the “current situation with  $i$  failures so far” and the “next situation with  $i + 1$  failures so far.” In addition,  $O'$  contains a set of auxiliary “goal-achieving” operators  $\{o_i^*\}_{i=0}^\kappa$  with

$$\text{pre}(o_i^*) = [G]_i \cup \{open_i\} \cup \bigcup_{j=i+1}^\kappa \{\neg open_j\},$$

$$\text{eff}(o_i^*) = \{ \langle \emptyset, \emptyset, \{open_i\} \rangle \}.$$

**Initial state and goal.** The  $P_0$ -part of the initial state  $\sigma'_0$  captures the sole initial state of  $\Pi$ , and for  $i > 0$ , the  $P_i$ -parts of the initial state are actually not important, and can be set arbitrarily. The auxiliary variable  $open_0$  is initially set to true, the rest of the variables  $open_i$  are initially set to false, and the goal of  $\Pi'$  is to negate  $open_0$ . That is,  $\sigma'_0 = [s_0]_0 \cup \{open_0\}$  and  $G' = \{\neg open_0\}$ .

For an illustration, consider a small and simplified variant of our running example in which there are only two locations,  $x$  and  $\text{not}(x)$ , the robot and a single spare tire are initially at  $x$ , and the goal is for the robot to be at  $\text{not}(x)$ . Movement of the robot either succeeds (which is the primary effect of that operator), or fails, with the robot staying at the original location with a flat tire. This planning task  $\Pi$  is encoded using propositions  $P = \{x, \text{noflat}, \text{spare}\}$  by the operators as in the Table 1a, initial state  $s_0 = \{x, \text{spare}, \text{noflat}\}$ , and goal  $G =$

	$o$	pre	eff	$\mathcal{F}$
(a)	move	$\{x, \text{noflat}\}$	$e_0 = \langle \emptyset, \{x\} \rangle$ $e_1 = \langle \emptyset, \{\text{noflat}\} \rangle$	$\mathcal{F}(e_0) = 0$ $\mathcal{F}(e_1) = 1$
	fix	$\{x, \text{spare}\}$	$e_0 = \langle \{\text{noflat}\}, \{\text{spare}\} \rangle$	$\mathcal{F}(e_0) = 0$

	$o$	pre	eff
(b)	move <sub>0</sub>	$\left\{ \begin{array}{l} x_0, \\ \text{noflat}_0, \\ \text{open}_0, \\ \neg \text{open}_1 \end{array} \right\}$	$\left\{ \begin{array}{l} \langle \emptyset, \emptyset, \{x_0\} \rangle \\ \langle \emptyset, \{\text{open}_1\}, \emptyset \rangle \\ \langle \emptyset, \emptyset, \{\text{noflat}_1\} \rangle \\ \langle \{x_0\}, \{x_1\}, \emptyset \rangle \\ \langle \{\neg x_0\}, \emptyset, \{x_1\} \rangle \\ \langle \{\text{spare}_0\}, \{\text{spare}_1\}, \emptyset \rangle \\ \langle \{\neg \text{spare}_0\}, \emptyset, \{\text{spare}_1\} \rangle \end{array} \right\}$
	move <sub>1</sub>	$\{x_1, \text{noflat}_1, \text{open}_1\}$	$\{\langle \emptyset, \emptyset, \{x_1\} \rangle\}$
	fix <sub>0</sub>	$\{x_0, \text{spare}_0, \text{open}_0, \neg \text{open}_1\}$	$\{\langle \emptyset, \{\text{noflat}_0\}, \{\text{spare}_0\} \rangle\}$
	fix <sub>1</sub>	$\{x_1, \text{spare}_1, \text{open}_1\}$	$\{\langle \emptyset, \{\text{noflat}_1\}, \{\text{spare}_1\} \rangle\}$
	$o_0^*$	$\{\neg x_0, \text{open}_0, \neg \text{open}_1\}$	$\{\langle \emptyset, \emptyset, \{\text{open}_0\} \rangle\}$
	$o_1^*$	$\{\neg x_1, \text{open}_1\}$	$\{\langle \emptyset, \emptyset, \{\text{open}_1\} \rangle\}$

Table 1: Operators from the compilation example.

$\{\neg x\}$ . The compilation  $\Pi' = \langle P', O', \sigma'_0, G' \rangle$  of the FT-planning task  $\langle \Pi, \mathcal{F}, 1 \rangle$  is defined over propositions  $P' = \bigcup_{i \in \{0,1\}} \{x_i, \text{noflat}_i, \text{spare}_i, \text{open}_i\}$ , and operators as in Table 1b. The initial state is  $\sigma'_0 = \{x_0, \text{spare}_0, \text{noflat}_0, \text{open}_0\}$ , and the goal is  $G' = \{\neg \text{open}_0\}$ . It is not hard to verify that  $\pi = \langle \text{move}_0, \text{fix}_1, \text{move}_1, o_1^*, o_0^* \rangle$  is (the only) plan for the classical planning task  $\Pi'$ , and that the respective contingent plan for  $\Pi^{(\mathcal{F},1)}$  can be decoded from  $\pi$  in linear time.

In the spirit of this example, we have generated a set of tasks in which a robot should move from the bottom-left to the top-right corner of a 4-connected grid, in which some of the edges are “safe,” with moves along them being deterministic, while other edges are “unsafe,” with moves on them either succeeding (which is the primary, expected effect of that operator) or resulting in the robot getting a flat and staying where it was. A limited number of spare tires are available on some nodes of the grid. The six sets of five tasks each correspond to  $5 \times 5$  and  $7 \times 7$  grids, with each edge of the grid being independently marked as safe with probability  $p \in \{0.1, 0.2, 0.5\}$ , and 10 spare tires, independently positioned on the grid nodes at random.

The runtimes of different approaches on these tasks are depicted in Table 2. The three approaches we examined were (col. 2) contingent planning with Contingent-FF (Hoffmann and Brafman 2005); (col. 3-6) FT-planning with Contingent-FF over  $(\mathcal{F}, \kappa)$ -reformulations,  $\kappa \in \{0, 1, 2, 4\}$ , and (col. 7-10) FT-planning with Fast Downward’s GBFS with FF heuristic over the classical planning reductions of the  $(\mathcal{F}, \kappa)$ -reformulations. Each task/planner was given a 10 minute time limit; cases in which the planner neither solved the task nor proved it unsolvable within the time bound are marked with ‘-’. If a planner solved a task within the time bound, then the respective entry in the table is shaded.

As Table 2 shows, all but one of these tasks were proven by Contingent-FF to have no strong contingent plans (and cyclic contingent plans are also of no help in this domain), while all (effectively classical) FT-planning tasks with  $\kappa = 0$  were easily solved by both Contingent-FF and Fast Down-

		CFF( $\Pi^{(\mathcal{F}, \kappa)}$ )				FD( $\Pi'$ )			
task		0	1	2	4	0	1	2	4
$5 \times 5$ (0.1)	0.08	0.00	-	-	-	0.00	0.10	0.02	0.04
	0.08	0.00	-	-	-	0.00	0.31	-	0.13
	0.08	0.00	-	-	-	0.00	0.10	-	0.13
	0.08	0.00	-	-	-	0.00	0.00	0.01	0.03
	0.08	0.00	4.99	-	-	0.00	0.01	0.01	0.03
	0.08	0.00	0.13	-	-	0.00	0.01	0.03	0.06
$5 \times 5$ (0.2)	0.08	0.00	-	-	-	0.00	0.19	0.02	0.04
	0.08	0.00	-	-	-	0.00	0.01	0.01	0.03
	0.08	0.00	-	-	-	0.00	0.00	0.01	0.03
	0.08	0.00	1.28	2.59	-	0.00	0.70	-	0.21
	0.08	0.00	-	-	-	0.00	7.50	0.02	0.04
	0.08	0.00	0.55	0.59	0.79	0.00	0.01	0.18	106.53
$5 \times 5$ (0.5)	0.08	0.00	0.12	5.04	5.43	0.00	0.01	0.02	0.04
	0.08	0.00	-	-	-	0.00	0.01	310.59	-
	0.07	0.00	-	-	-	0.00	0.01	0.02	0.04
	0.12	0.00	-	-	-	0.00	0.02	0.03	0.06
	0.13	0.00	2.10	-	-	0.00	1.67	0.04	0.07
	0.13	0.00	-	-	-	0.00	0.21	0.03	0.07
$7 \times 7$ (0.1)	0.13	0.00	-	-	-	0.00	0.02	0.03	0.06
	0.13	0.00	-	-	-	0.00	0.09	0.04	0.07
	0.13	0.00	-	-	-	0.00	27.32	0.04	0.08
	0.13	0.00	-	-	-	0.00	0.01	0.03	0.06
	0.13	0.00	-	-	-	0.00	0.02	0.03	0.06
	0.13	0.00	-	-	-	0.00	0.01	0.03	0.06
$7 \times 7$ (0.2)	0.13	0.00	-	-	-	0.00	5.96	0.04	0.07
	0.13	0.00	-	-	-	0.00	0.38	0.05	0.09
	0.13	0.00	3.32	4.13	-	0.00	0.04	0.63	11.56
	0.13	0.00	-	-	-	0.00	0.31	38.86	-
	0.13	0.00	0.14	0.15	0.15	0.00	0.01	0.03	0.06
	0.13	0.00	-	-	-	0.00	0.89	17.37	1.25

Table 2: Planner runtimes on different formulations of FT-planning tasks in the spirit of our example.

ward. For us, of course, the interesting part was in between these two extremes, and both Contingent-FF and Fast Downward found non-trivial  $\kappa$ -plans for numerous tasks here. In terms of performance, compiling the contingent  $(\mathcal{F}, \kappa)$ -reformulations to classical planning strictly dominated solving the former directly, in terms of the coverage of both solvable and unsolvable FT-planning tasks. In sum, the direction of compiling FT-planning tasks to classical planning appears promising, and clearly deserves further investigation.

In Lemma 5 below, we now formalize the properties of FT-PLAN-1- $\kappa$  that are exploited by the compilation of its fragment above. In particular, this lemma allows for extending this compilation scheme to arbitrary *fixed* bounds on the number of non-deterministic effects per operator, as well as to arbitrary normative 1-primary exception models.

**Lemma 4** *Let  $\langle \Pi = \langle P, O, s_0, G \rangle, \mathcal{F}, \kappa \rangle$  be an FT-planning task with a 1-primary model  $\mathcal{F}$ , and  $\max_{o \in O} \text{eff}(o) = b$ . If  $\pi$  is an irreducible contingent plan for  $\Pi^{(\mathcal{F}, \kappa)}$ , then there exists a set of policies  $\pi_0, \dots, \pi_m$  over  $S^{(\mathcal{F}, \kappa)}$  such that*

- (1)  $\pi_0$  is an empty policy,  $\pi_m = \pi$ , and each  $\pi_i$  extends  $\pi_{i-1}$  by prescribing an action for a single additional state of  $\Pi^{(\mathcal{F}, \kappa)}$  such that the execution tree  $T_{\pi_{i-1}}$  is a proper sub-tree of  $T_{\pi_i}$ , and
- (2) for  $0 \leq i \leq m$  and  $0 \leq j \leq \kappa$ ,  $\pi_i$  induces at most  $b$  executions  $\rho$  that do not achieve the goal within  $T_{\pi_i}$  and have  $\mathcal{F}(\rho) = j$ .

The proof is as follows. Let  $\pi$  be an irreducible contingent plan for the  $(\mathcal{F}, \kappa)$ -reformulation of  $\langle \Pi, \mathcal{F}, \kappa \rangle$  as in the claim, and let  $\{(s_1, k_1), \dots, (s_m, k_m)\}$  be a relabeling of the nodes  $S_{\pi}^{(\mathcal{F}, \kappa)}$  consistently with the order in which

they are expanded by a depth-first traversal of  $T_\pi$ , with the “depth” of a node  $(s, k)$  being given by  $k$ . Given that, let a sequence of policies  $M = \pi_0, \dots, \pi_m$  be defined as

$$\pi_i(s_j, k_j) = \begin{cases} \pi(s_j, k_j), & j \leq i \\ \text{undefined}, & j > i \end{cases}$$

It is immediate that  $M$  satisfies condition (1) of the lemma, and so what remains to be shown is condition (2). The proof is by induction on  $i$ . For  $i = 0$ , the condition is trivially satisfied since  $\pi_0$  is empty. Assuming that the condition is satisfied for  $i - 1 \geq 0$ , the proof for  $i$  is as follows. By the DFS construction of  $M$ , we have  $S_{\pi_i}^{(\mathcal{F}, \kappa)} = S_{\pi_{i-1}}^{(\mathcal{F}, \kappa)} \cup \{(s_i, k_i)\}$ , where  $(s_i, k_i)$  is a non-goal leaf node in  $T_{\pi_{i-1}}$ . Furthermore, for all other non-goal leaf nodes  $(s_j, k_j)$  of  $T_{\pi_{i-1}}$ , it holds that  $k_j \leq k_i$ , or otherwise DFS would expand  $(s_j, k_j)$  prior to  $(s_i, k_i)$ . Given that, consider the extension of  $T_{\pi_{i-1}}$  to  $T_{\pi_i}$  by  $\pi(s_i, k_i)$ .

By the definition of exception models, for each  $k < k_i$ , the number of executions  $\rho$  of  $\pi_i$  that do not achieve the goal within  $T_{\pi_i}$  and have  $\mathcal{F}(\rho) = k$  is the same as for  $\pi_{i-1}$ , and this because the number of “exceptions so far” cannot decrease with the progress of the execution. For  $k = k_i$ , since  $\mathcal{F}$  is 1-primary,  $\pi_i$  replaces a single execution  $\rho$  of  $\pi_{i-1}$  that does not achieve the goal within  $T_{\pi_{i-1}}$  and has  $\mathcal{F}(\rho) = k_i$ , with at most one such execution, namely the one that extends  $\rho$  with the sole primary effect of  $\pi(s_i, k_i)$ . Finally, for all  $k > k_i$ , there are no executions of  $\pi_{i-1}$  that do not achieve the goal within  $T_{\pi_{i-1}}$  and have  $\mathcal{F}(\rho) = k$ , and thus there are at most  $b$  such executions of  $\pi_i$ .

Given an FT-planning task  $\langle \Pi = \langle P, O, s_0, G \rangle, \mathcal{F}, \kappa \rangle$  with a normative 1-primary model  $\mathcal{F}$  and  $\max_{o \in O} \text{eff}(o) = O(1)$ , a polynomial-time, sound, and complete compilation of  $\langle \Pi, \mathcal{F}, \kappa \rangle$  to a classical planning task  $\Pi' = \langle P', O', \sigma'_0, G' \rangle$  is specified below. For ease of presentation, for each operator  $o \in O(s)$ , if  $\text{eff}(o) = \{e_0, e_1, \dots, e_{b'}\}$ , then  $\mathcal{F}(e_0) = 0$ . The set of propositions  $P'$  contains  $\kappa(b-1)+1$  replicas of propositions  $P$ , as well as  $\kappa(b-1)+1$  auxiliary propositions *open*, denoted as

$$P' = P_{0,0} \cup \{\text{open}_{0,0}\} \cup \bigcup_{\substack{1 \leq i \leq \kappa, \\ 1 \leq j \leq b-1}} P_{i,j} \cup \{\text{open}_{i,j}\}.$$

The set of operators  $O'$  contains  $\kappa(b-1)+1$  sets of operators  $O_{0,0}, O_{1,1}, \dots, O_{1,b-1}, \dots, O_{\kappa,1}, \dots, O_{\kappa,b-1}$ , with  $O_{i,j} = \{o_{i,j} \mid o \in O\}$ . For each  $o \in O$ , the precondition of the operator  $o_{i,j}$  is

$$\begin{aligned} \text{pre}(o_{i,j}) = & [\text{pre}(o)]_{i,j} \cup \{\text{open}_{i,j}\} \cup \bigcup_{x=j+1}^{b-1} \{\neg \text{open}_{i,x}\} \\ & \cup \bigcup_{y=i+1}^{\kappa} \bigcup_{x=1}^{b-1} \{\neg \text{open}_{y,x}\}. \end{aligned}$$

If  $\text{eff}(o) = \{e_0, e_1, \dots, e_{b'}\}$ , then

$$\begin{aligned} \text{eff}_{i,j} = & \{\langle \emptyset, [\text{add}(e_0)]_{i,j}, [\text{del}(e_0)]_{i,j} \rangle \cup \\ & \bigcup_{\substack{1 \leq x \leq b', \\ i+\mathcal{F}(e_x) \leq \kappa}} \left\{ \langle \emptyset, \{\text{open}_{i+\mathcal{F}(e_x),x}\}, \emptyset \rangle, \right. \\ & \left. \langle \emptyset, [\text{add}(e_1)]_{i+\mathcal{F}(e_x),x}, [\text{del}(e_1)]_{i+\mathcal{F}(e_x),x} \rangle \right\} \\ & \cup \Phi_{i,j,x} \end{aligned}$$

where

$$\Phi_{i,j,x} = \bigcup_{p \notin \text{add}(e_x) \cup \text{del}(e_x)} \left\{ \langle \{p_{i,j}\}, \{p_{i+\mathcal{F}(e_x),x}\}, \emptyset \rangle, \langle \{\neg p_{i,j}\}, \emptyset, \{p_{i+\mathcal{F}(e_x),x}\} \rangle \right\}.$$

Likewise,  $O'$  contains a set of auxiliary operators  $\{o_{0,0}^*, o_{1,1}^*, \dots, o_{1,b-1}^*, \dots, o_{\kappa,1}^*, \dots, o_{\kappa,b-1}^*\}$ , with

$$\begin{aligned} \text{pre}(o_{i,j}^*) = & [G]_{i,j} \cup \{\text{open}_{i,j}\} \cup \bigcup_{x=j+1}^{b-1} \{\neg \text{open}_{i,x}\} \\ & \cup \bigcup_{y=i+1}^{\kappa} \bigcup_{x=1}^{b-1} \{\neg \text{open}_{y,x}\}, \\ \text{eff}(o_i^*) = & \{\langle \emptyset, \emptyset, \{\text{open}_{i,j}\} \rangle\}. \end{aligned}$$

Finally, as in the basic case, the initial state and goal are specified as  $\sigma'_0 = [s_0]_{0,0} \cup \{\text{open}_{0,0}\}$  and  $G' = \{\neg \text{open}_{0,0}\}$ .

**Theorem 5** *Let  $\langle \Pi = \langle \mathcal{F}, \kappa \rangle$  be an FT-planning task with a 1-primary model  $\mathcal{F}$ , and  $\Pi'$  be the compilation of  $\Pi$ . There is a bijective, efficiently computable mapping between the irreducible plans for  $\Pi^{(\mathcal{F}, \kappa)}$  and those for  $\Pi'$ .*

## Summary

We studied computational properties of fault tolerant planning, a simple and natural planning formalism that interpolates between contingent and classical planning. We showed that an important spot along this interpolation exhibits attractive worst-case time complexity, and for most, can be efficiently compiled to classical planning in a sound, complete, and practicable manner.

The palette of possible (and impossible) extensions to FT-planning that call for investigation is wide. For instance, it is possibly more natural in some contexts to assume bounds on the number of failures per operator, and not on the number of failures overall, as we do here. It is easy to show that this type of assumption-based planning is also in PSPACE, but our more practicable results here (NP-membership for problems with polynomially-long executions and the specific compilation scheme to classical planning) do not apply there. Also, for simplicity, throughout the paper we assumed a single (aka fully known a priori) initial state. It is not hard to verify, however, that our PSPACE-membership, NP-membership, and compilation results can be straightforwardly extended to arbitrary, polynomial, and fixed numbers of possible initial states, respectively, as long as listing these possible initial states does not introduce further complexity. Finally, we believe that FT-planning to classical planning compilation can be substantially stratified by exploiting the structure of the FT-planning tasks, similarly to the way the structure of the tasks is exploited in recent compilations from conformant to classical planning (Palacios and Geffner 2009).

**Acknowledgments.** This work was partially supported by the EOARD grant FA8655-12-1-2096, and the ISF grant 1045/12.

## References

- Albore, A., and Bertoli, P. 2004. Generating safe assumption-based plans for partially observable, nondeterministic domains. In *AAAI*, 495–500.
- Albore, A., and Bertoli, P. 2006. Safe LTL assumption-based planning. In *ICAPS*, 193–202.
- Albore, A., and Geffner, H. 2009. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*.
- Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.
- Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*, 1936–1941.
- Bonet, B. 2010. Conformant plans and beyond: Principles and complexity. *AIJ* 174(3-4):245–269.
- Brafman, R. I., and Shani, G. 2012a. A multi-path compilation approach to contingent planning. In *AAAI*.
- Brafman, R. I., and Shani, G. 2012b. Replanning in domains with partial information and sensing actions. *JAIR* 45:565–600.
- Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AIJ* 69(1-2):165–204.
- Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AIJ* 147(1-2):35–84.
- Davis-Mendelow, S.; Baier, J. A.; and McIlraith, S. A. 2012. Making reasonable assumptions to plan with incomplete information. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*.
- Domshlak, C.; Even-Zur, Z.; Golany, Y.; Karpas, E.; and Nus, Y. 2011. Command and control training centers: Computer generated forces meet classical planning. In *ICAPS Workshop on Scheduling and Planning Applications (SPARK)*.
- Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ* 2:189–208.
- Georgeff, M., and Lansky, A. L. 1986. Procedural knowledge. *Proceedings of IEEE* 74(10):1383–1398.
- Giunchiglia, F.; Spalazzi, L.; and Traverso, P. 1994. Planning with failure. In *AIPS*, 74–79.
- Göbelbecker, M.; Gretton, C.; and Dearden, R. 2011. A switching planner for combined task and observation planning. In *AAAI*.
- Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.
- Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS*, 71–80.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.
- Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2004. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *ICAPS*, 335–344.
- Kissmann, P., and Edelkamp, S. 2012. Improving cost-optimal domain-independent symbolic planning. In *AAAI*.
- Littman, M. 1997. Probabilistic propositional planning: representations and complexity. In *AAAI*, 748754.
- Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.
- Pistore, M., and Vardi, M. Y. 2007. The planning spectrum - One, two, three, infinity. *JAIR* 30:101–132.
- Puterman, M. 1994. *Markov Decision Processes*. Wiley.
- Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *AIJ* 170(12-13):1031–1080.
- Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*, 345–354.
- Taig, R., and Brafman, R. I. 2013. Compiling conformant probabilistic planning into classical planning. In *ICAPS*.
- Talamadupula, K.; Benton, J.; Schermerhorn, P. W.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a closed world planner with an open world robot: A case study. In *AAAI*.
- Williams, B. C.; Ingham, M.; Chung, S. H.; and Elliott, P. H. 2003. Procedural knowledge. *Proceedings of IEEE* 9:212237.
- Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.

# Monte-Carlo Tree Search: To MC or to DP?

Zohar Feldman and Carmel Domshlak<sup>1</sup>

**Abstract.** State-of-the-art Monte-Carlo tree search algorithms can be parametrized with any of the two information updating procedures: MC-backup and DP-backup. The dynamics of these two procedures is very different, and so far, their relative pros and cons have been poorly understood. Formally analyzing the dependency of MC- and DP-backups on various MDP parameters, we reveal numerous important issues that get hidden by the worst-case bounds on the algorithm performance, and reconfirm these findings by a systematic experimental test.

## 1 INTRODUCTION

Markov decision processes (MDPs) is a standard model for planning under uncertainty [17]. An MDP  $\langle S, A, \mathbb{P}, R \rangle$  is defined by a set of states  $S$ , a set of state transforming actions  $A$ , a stochastic transition function  $\mathbb{P} : S \times A \times S \rightarrow [0, 1]$ , and a reward function  $R : S \times A \times S \rightarrow \mathbb{R}$ . The states are fully observable and, in the finite horizon setting considered here, the rewards are accumulated over some predefined number of steps  $H$ . The objective of planning in MDPs is to sequentially choose actions so as to maximize the accumulated reward. The representation of large-scale MDPs can be either declarative or generative, but anyway concise, and allowing for simulated execution of all feasible action sequences, from any state of the MDP. In *online MDP planning*, the agent focuses on its current state  $s_0$  only, deliberates about the set of possible policies from that state onwards and, when interrupted, chooses what action to perform next. In formal analysis of algorithms for online MDP planning, the quality of the action  $a$ , chosen for  $s_0$  with  $H$  steps-to-go, is assessed in terms of the induced “simple regret”, capturing the performance loss that results from taking  $a$  and then following an optimal policy  $\pi^*$  for the remaining  $H - 1$  steps, instead of following  $\pi^*$  from the beginning [4].

Many popular algorithms for online MDP planning constitute what is called *Monte-Carlo tree search (MCTS)* [21, 16, 15, 7, 6, 19, 22, 14, 10, 12], and adaptations of some of these algorithms are also popular in other settings of sequential decision making, including those with partial state observability and adversarial effects [13, 20, 2, 8, 3]. At a high level, all MCTS algorithms explore the state-space region around  $s_0$  by iteratively (i) simulating an action/state trajectory from  $s_0$ , and (ii) using the outcome of that trajectory to update various action-value estimates related to the state-space region of interest, as well as to update the estimate of what action should be best applied at state  $s_0$ . In that respect, specific MCTS algorithms differ both in their trajectory rollout strategies, as well as in their rollout-based update strategies.

Recent work substantially advanced our understanding of how the performance of MCTS depends on the specifics of the rollout strategy, as well as on the choice of what pieces of information should be

updated based on a given rollout [15, 7, 9]. Recently, however, Keller & Helmert [14] demonstrated empirically that the performance of MCTS also depends to a large extent on *how* the respective updates are being performed. Prior to the work of Keller & Helmert, updates in MCTS algorithms were all based on *MC-backups*, that is, sample averaging updates of the selected action-value estimates. Keller & Helmert showed that modifying a standard MCTS algorithm (such as UCT [15]), by replacing its MC-backups with dynamic programming estimates propagation *a la* Bellman backups in value iteration [1], can substantially affect the performance, and, at least in their experiments, *typically in favor of DP-backups*. Later on, Feldman & Domshlak [12] showed that switching from MC-backups to such *DP-backups* preserves the order-of-magnitude convergence rates of MCTS instances that guarantee exponential rate performance improvement (such as BRUE [9]), and can even allow for proving *somewhat better convergence bounds*. Still, the relative pros and cons of MC- and DP-backups have not been systematically studied so far, and thus are still poorly understood.

This is precisely our contribution in this paper: Using BRUE and MaxBRUE, a pair of state-of-the-art MCTS algorithms that, *ceteris paribus*, use MC-backups and DP-backups, respectively, we study the dynamics of MC- and DP-backups both formally and empirically. Starting with establishing a pair of comparable worst-case bounds on the convergence rates of these two algorithms, we use the analysis behind these bounds to examine specific dependencies of the two algorithms on various MDP parameters, namely the state and action branching factors, the shape of the reward function, and the entropy of the transition function. To our knowledge, our analysis is first of its kind, and it reveals numerous important issues that get hidden by the worst-case bounds due to certain deficiencies in the formal worst-case analysis of MC-backups. In particular, it suggests that MC-backups are less sensitive than DP-backups to the state branching factor, especially when the transition function concentrates on a small number of outcomes, as it is typically the case in practical applications. The various aspects of the analysis are then put on a systematic experimental test, which reconfirms its key findings.

## 2 BACKGROUND

In what follows, we adopt the notation and pseudo-code convention from [12]. In particular, when considering an MDP  $\langle S, A, \mathbb{P}, R \rangle$ , its state and action branching factors are respectively denoted by  $K = \max_s |A(s)|$  and  $B = \max_{s,a} |\{s' \mid \mathbb{P}(s'|s, a) > 0\}|$ ,  $s(h)$  denotes state  $s \in S$  with  $h$  steps-to-go, and  $A(s) \subseteq A$  denotes the actions applicable in state  $s$ . Some auxiliary notation: The operation of drawing a sample from a distribution  $\mathcal{D}$  over set  $\aleph$  is denoted by  $\sim \mathcal{D}[\aleph]$ ,  $\mathcal{U}$  denotes uniform distribution, and  $\llbracket n \rrbracket$  for  $n \in \mathbb{N}$  denotes the set  $\{1, \dots, n\}$ . For a sequence of tuples  $\rho$ ,  $\rho[i]$  denotes the  $i$ -th tuple along  $\rho$ , and  $\rho[i].x$  denotes the value of the field  $x$  in that tuple.

<sup>1</sup> Technion, Israel, email: {zoharf@tx,dcarmel@ie}.technion.ac.il

<pre> MCTS: [input: <math>\langle S, A, Tr, R \rangle; s_0 \in S</math>] while time permits do   <math>\rho \leftarrow \text{ROLLOUT}</math>   UPDATE(<math>\rho</math>) return <math>\arg \max_a \hat{Q}(s_0 \langle H \rangle, a)</math>  procedure ROLLOUT   <math>\rho \leftarrow \langle \rangle; s \leftarrow s_0; d \leftarrow 0</math>   while not STOP-ROLLOUT(<math>\rho</math>) do     <math>a \leftarrow \text{ROLLOUT-ACTION}(s \langle H-d \rangle)</math>     <math>s' \leftarrow \text{ROLLOUT-OUTCOME}(s \langle H-d \rangle, a)</math>     <math>r \leftarrow R(s, a, s')</math>     <math>\rho[t] \leftarrow \langle s, a, r, s' \rangle</math>     <math>s \leftarrow s'; d \leftarrow d+1</math>   return <math>\rho</math> </pre> <p>(a) MCTS</p>	<pre> procedure UPDATE(<math>\rho</math>)   for <math>d \leftarrow  \rho , \dots, 1</math> do     <math>h \leftarrow H-d</math>     <math>\langle s, a, r, s' \rangle \leftarrow \rho[d]</math>     <math>n(s \langle h \rangle) \leftarrow n(s \langle h \rangle) + 1</math>     <math>n(s \langle h \rangle, a) \leftarrow n(s \langle h \rangle, a) + 1</math>     <math>n(s \langle h \rangle, a, s') \leftarrow n(s \langle h \rangle, a, s') + 1</math>     <math>\bar{r} \leftarrow r + \text{ESTIMATE}(s' \langle h-1 \rangle)</math>     MC-BACKUP(<math>s \langle h \rangle, a, \bar{r}</math>)    procedure ESTIMATE(<math>s \langle h \rangle</math>)     <math>\bar{r} \leftarrow 0</math>     for <math>d \leftarrow 0, \dots, h-1</math> do       <math>a \leftarrow \text{EST-ACTION}(s \langle h-d \rangle)</math>       <math>s' \leftarrow \text{EST-OUTCOME}(s \langle h-d \rangle, a)</math>       <math>\bar{r}_{d+1} \leftarrow R(s, a, s')</math>       <math>\bar{r} \leftarrow \bar{r} + \bar{r}_{d+1}</math>     <math>s \leftarrow s'</math>   return <math>\bar{r}</math>    procedure MC-BACKUP(<math>s \langle h \rangle, a, \bar{r}</math>)     <math>\hat{Q}(s \langle h \rangle, a) \leftarrow \frac{n(s \langle h \rangle, a)-1}{n(s \langle h \rangle, a)} \hat{Q}(s \langle h \rangle, a) + \frac{1}{n(s \langle h \rangle, a)} \bar{r}</math> </pre> <p>(b) MC</p>	<pre> procedure UPDATE(<math>\rho</math>)   for <math>d \leftarrow  \rho , \dots, 1</math> do     <math>h \leftarrow H-d</math>     <math>a \leftarrow \rho[d].a</math>     <math>s' \leftarrow \rho[d].s'</math>     <math>n(s \langle h \rangle) \leftarrow n(s \langle h \rangle) + 1</math>     <math>n(s \langle h \rangle, a) \leftarrow n(s \langle h \rangle, a) + 1</math>     <math>n(s \langle h \rangle, a, s') \leftarrow n(s \langle h \rangle, a, s') + 1</math>     <math>\hat{R}(s \langle h \rangle, a) = \hat{R}(s \langle h \rangle, a) + \rho[d].r</math>     DP-BACKUP(<math>s \langle h \rangle, a</math>)    procedure DP-BACKUP(<math>s \langle h \rangle, a</math>)     <math>\hat{Q}(s \langle h \rangle, a) \leftarrow \frac{\hat{R}(s \langle h \rangle, a)}{n(s \langle h \rangle, a)}</math>     <math>v \leftarrow 0</math>     for <math>s' \in \{s' \mid n(s \langle h \rangle, a, s') &gt; 0\}</math> do       <math>v \leftarrow v + \frac{n(s \langle h \rangle, a, s')}{n(s \langle h \rangle, a)} \max_{a'} \hat{Q}(s' \langle h-1 \rangle, a')</math>     <math>\hat{Q}(s \langle h \rangle, a) \leftarrow \hat{Q}(s \langle h \rangle, a) + v</math> </pre> <p>(c) DP</p>
---	--	--

**Figure 1.** (a) Monte-Carlo tree search general scheme, with “separation of concerns” versions of (b) MC-backup and (c) DP-backup updates

MCTS, a canonical scheme underlying various MCTS algorithms for online MDP planning, is depicted in Figure 1. MCTS explores the state space in the radius of  $H$  steps from the initial state  $s_0$  by iteratively issuing simulated ROLLOUTs from  $s_0$ . Each such rollout  $\rho$  comprises a sequence of simulated steps  $\langle s, a, r, s' \rangle$ , where  $s$  is a state,  $a$  is an action applicable in  $s$ ,  $r$  is an immediate reward collected from issuing the action  $a$ , and  $s'$  is the resulting state. Once generated, the rollout is used to UPDATE some variables of interest, typically including at least the action value estimators  $\hat{Q}(s \langle h \rangle, a)$  and the counters  $n(s \langle h \rangle, a)$  that record the number of times the corresponding estimators  $\hat{Q}(s \langle h \rangle, a)$  have been updated. Once interrupted, MCTS uses the information collected throughout the exploration to recommend an action to perform at state  $s_0$ .

Instances of MCTS vary mostly along their ROLLOUT-ACTION policies, prescribing the action to apply in the current state of the rollout; and their UPDATE strategies, specifying (i) which of the maintained variables should be updated based on the rollout, as well as (ii) how those variables should be updated. The “how” aspect of the MCTS UPDATE procedures is of our focus here. By decoupling between the decisions of what to update and how to update, the emphasized text in Figures 1b and 1c shows the respective subroutines for MC-backup and DP-backup, the two alternatives for “how to update” that are in use these days by various MCTS algorithms.

- MC-backups are based on the principle of averaging random variable samples: Given a new value sample  $\bar{r}$  for an action  $a$  at  $s \langle h \rangle$ ,  $\bar{r}$  updates the running sample average  $\hat{Q}(s \langle h \rangle, a)$ , either knowing or just assuming that this way  $\hat{Q}(s \langle h \rangle, a)$  will eventually converge to the true  $Q$ -value of  $a$  at  $s \langle h \rangle$ .
- DP-backups implement dynamic programming style estimates propagation, resembling Bellman backups in value iteration. With DP-backups, action value estimates  $\hat{Q}(s \langle h \rangle, a)$  are updated by the weighted sum of the value estimates of the empirically best actions at the outcomes of  $a$  (discovered so far), with the weights being induced by the gradually learned parameters of the MDP’s stochastic transition function.

Earlier MCTS algorithms, such as flat MC,  $\varepsilon$ -greedy, UCT, and their numerous variations [3], all reflected rather directly the algorithms for reinforcement learning-while-acting in multi-armed bandit

problems (MAB) [18]: Given a rollout  $\rho$ , update (the selected) action value estimates “by  $\rho$ ”, that is, by the actual rewards obtained along the rollout. Recent works on MCTS algorithms for online MDP planning examined the important differences between the (single state) MABs and (multi-state) general MDPs, leading to what was baptized as the principle of separation of concerns [9]: Instead of updating “by  $\rho$ ”, update (the selected) action value estimates “along the trajectory of  $\rho$ ” by some information that goes beyond, and possibly even has nothing to do with, the specific rewards achieved by  $\rho$ . In particular, one can use one of the following.

1. MC-updates along additional rollouts, issued from the states along  $\rho$  according to a special, update-oriented, “estimation” policy. Such an UPDATE procedure in particular gives rise to the BRUE algorithm [9], and it is depicted in Figure 1b, together with a general template for its ESTIMATE policy.
2. DP-updates along the trajectory of  $\rho$ , as depicted in Figure 1c. This procedure in particular gives rise to the MaxUCT [14] and MaxBRUE [12] algorithms.

### 3 WORST-CASE GUARANTEES VS. REALISTIC EXPECTATIONS

Our comparison between MC-backups and DP-backups in MCTS is carried through two particular MCTS algorithms, BRUE [9] and MaxBRUE [12], which guarantee exponential-rate reduction of simple regret. The only difference between BRUE and MaxBRUE is that the former employs MC-backups while the latter employs DP-backups. Hence, for ease of presentation, in what follows we refer to these two algorithms as MC and DP, respectively. Both MC and DP use uniform sampling for ROLLOUT-ACTION, and both use the same ROLLOUT-OUTCOME that samples the provided generative model of the action’s transition function. With UPDATE as in Figure 1c, that basically concludes the definition of DP. The UPDATE procedure of MC in Figure 1b, and in particular, its ESTIMATE subroutine, needs one more choice to be made.

In analogy to DP-backup that propagates the value of the empirically best actions, MC in ESTIMATE makes the estimation rollouts along the empirically best actions (selected by EST-ACTION). Thus,



in particular, no implementation choices are left open here. In contrast, for outcome selection along the estimation rollouts, two options are plausible, and both are viable in terms of consistency and performance guarantees. One option is to use the generative model of the action's transition function, same as in ROLLOUT-OUTCOME, while another option is to estimate the transition probabilities, similarly to DP, and draw samples from that empirical distribution.

The advantage of the second scheme is that the number of “oracle calls” to the generative model is similar to that of DP. In contrast, the first scheme performs a factor of  $\frac{H}{2}$  more calls to generative model. In applications where such oracle calls are expensive, due to, e.g., a need to simulate a complex physical model, this can be an important argument for using the second scheme. However, the advantage of the first scheme is that it is not affected by the errors in the estimation of the transition probabilities. In what follows, whenever we need to distinguish between the two options, we will use  $MC_M$  to refer to MC with EST-OUTCOME using the generative model, and  $MC_P$  to refer to MC with EST-OUTCOME using the estimated transition probabilities.

As we already mentioned, both DP and MC have been recently proven to reduce simple regret at exponential rate. However, the corresponding statements of the formal bounds in [9] and [12] are somewhat involved, and this complicates the comparative analysis and discussion of DP and MC that we want to make here. Propositions 1 and 2 below provide much more accessible formal bounds on the performance of DP and MC, simplified by assuming  $B, K \gg 0$ , which allows keeping track only of the highest order factors of  $B$  and  $K$ , and replacing uniform ROLLOUT-ACTION with round-robin, which is equivalent in expectation but allows for simplifying the bounds further. We note that, while the bound for DP in Proposition 1 is qualitatively similar to this in [12], the bound for MC in Proposition 2 actually improves over the result in [9]. The proofs are delegated to a technical report [11].

**Proposition 1** *Let  $\pi^B(s_0\langle H \rangle)$  be the action recommendation of DP after applying  $n$  iterations. Then,*

$$\begin{aligned} & \mathbb{P} \left\{ Q(s_0\langle H \rangle, \pi^*(s_0\langle H \rangle)) - Q(s_0\langle H \rangle, \pi^B(s_0\langle H \rangle)) \geq \delta \right\} \\ & \leq K(2BK)^{H-1} e^{-\frac{\delta^2 n}{4K(BK)^{H-1}H^2}} \end{aligned}$$

**Proposition 2** *Let  $\pi^B(s_0\langle H \rangle)$  be the action recommendation of MC after applying  $n$  iterations. Then,*

$$\begin{aligned} & \mathbb{P} \left\{ Q(s_0\langle H \rangle, \pi^*(s_0\langle H \rangle)) - Q(s_0\langle H \rangle, \pi^B(s_0\langle H \rangle)) \geq \delta \right\} \\ & \leq \left( \frac{6B^2K}{\Delta^2} \right)^{H-1} (9BK)^{\frac{1}{2}(H-1)^2} (H-1)!^2 e^{-\frac{\Delta^2 n}{2K(9BK)^{H-1}H^2}} \end{aligned}$$

Roughly speaking, the exponents in the bounds in Propositions 1 and 2 capture the reduction rate of the simple regret, while the multiplicative factors capture the length of the “cooling periods” after which the respective bounds become meaningful. In that respect, the convergence rates of DP and MC appear to be rather comparable, excluding numerical constants, while the “cooling period” of MC appears to be much longer than that of DP. The latter suggests, even if only informally, that the empirical performance of DP should be expected to be more attractive than the empirical performance of MC. However, a deeper inspection of MC and DP below suggests a different perspective on the relative attractiveness of these two algorithms, and more generally, on the relative attractiveness of MC-backups and DP-backups in MCTS.

First, in [9] it was shown that the formal guarantees of MC can be improved by basing the action value estimators only on a fraction  $\alpha$  of the most recent samples. This enhancement was referred in [9] as “learning by forgetting”. For some specific values of  $\alpha$  convenient for our discussion here, the bound for MC from Proposition 1 translates to a bound for the “learning by forgetting”  $MC(\alpha)$  as in Proposition 3 below.

**Proposition 3** *Let  $\pi^B(s_0\langle H \rangle)$  be the action recommendation of  $MC(\alpha)$  after applying  $n$  iterations with a steps-to-go-dependent averaging fraction  $\alpha_h \sim \frac{1}{(9BK)^{h-1}}$ . Then, we have*

$$\begin{aligned} & \mathbb{P} \left\{ Q(s_0\langle H \rangle, \pi^*(s_0\langle H \rangle)) - Q(s_0\langle H \rangle, \pi^B(s_0\langle H \rangle)) \geq \delta \right\} \\ & \leq \left( \frac{40BK}{\delta^2} \right)^{H-1} (H-1)!^2 e^{-\frac{\delta^2 n}{2K(9BK)^{H-1}H^2}} \end{aligned}$$

As it appears, the worst-case cooling period of  $MC(\alpha)$  improves on that of MC, and this seems to come at no cost in terms of the convergence rate, expressed by the exponent. However, as we explain below, it seems that this improvement of the bound in Proposition 3 should be attributed mostly to the looseness of the bound for the standard setup of MC, and much less to the actual improvement of the performance measures. Indeed, adopting “learning by forgetting” leads to only minor empirical improvement, if at all.<sup>2</sup> Nevertheless, in comparison to DP, the cooling period length of  $MC(\alpha)$  has stronger dependency on the accuracy level  $\delta$  and the horizon  $H$ .

At this point, two things should be noted with regards to the above formal bounds. First, by definition, formal bounds capture the worst-case settings of the MDP parameters, that is, uniform transition probability functions, tree-structured state space, etc. As such, the bounds tend to blur certain advantages of one algorithm over another in solving MDPs with some specific (and possibly expected in practice) characteristics. Second, due to conceptual differences between the dynamics of MC- and DP-backups, derivation of the bounds in Propositions 1 and 2 is based on two very different types of analysis. Hence, unlike what often happens for conceptually close techniques [5, 9], the value of formal bounds as indicators for the relative attractiveness of MC and DP is questionable. Having these two reservations in mind, in what follows we provide a more conceptual (aka less mathematically specific) comparative analysis of MC and DP by exploring several key features of MDP models, and reasoning about the (possibly different) effects of each of these features on the performance of the two algorithms.

**Branching factors  $B$  and  $K$  (The size of the problem)** In both Proposition 1 and Proposition 2, the basis of the analysis that gives rise to the formal guarantees is the fact that identifying the optimal action  $a^*$  at the root node  $s_0\langle H \rangle$  requires that

- (1) the value of  $a^*$  at  $s_0\langle H \rangle$  is not too underestimated, and that
- (2) the values of all the other, sub-optimal actions at  $s_0\langle H \rangle$  are not too overestimated.

Both MC and DP ensure that these accuracy requirements are met, yet they differ in the way that these requirements recursively translate into requirements from the descendants of  $s_0\langle H \rangle$ .

In DP, to ensure that a sub-optimal action  $a$  is not too overestimated, all the applicable actions in all of the outcome states of  $a$  must not be too overestimated. Thus, the accuracy of estimating  $a$

<sup>2</sup> This was observed both in our experiments for this work, as well as in [9].



sub-optimal action  $a$  in DP translates to accuracy requirements being posed to all the possible  $BK$  immediate action successors of  $a$ . In contrast, in MC, the likelihood that a sub-optimal action  $a$  will be too overestimated is negligible, and this because the expected value of the samples that induce the estimate of  $a$  is upper-bounded by the true value of  $a$ , regardless of the estimates of the action successors of  $a$ . Thus, the accuracy of estimating a sub-optimal action  $a$  in MC translates to no accuracy requirements from the successors of  $a$ . In sum, in terms of “not overestimating sub-optimal actions”, MC-backups seem to be clearly preferred to DP-backups.

Examining now the requirement of not underestimating the optimal action  $a^*$  too much, meeting this requirement in DP requires that the optimal actions at all of the outcome states of  $a^*$  are not too underestimated. Indeed, if the latter holds, then the maximal action values propagated to  $a^*$  from its outcome states are also not too underestimating. Thus, the requirement of “not too underestimating the optimal action”  $a^*$  at  $s_0(H)$  translates in DP into  $B$  similar requirements being posed to all of the state successors of  $s_0(H)$  via  $a^*$ .

When it comes to MC, the picture is somewhat more complicated. Not underestimating the optimal action  $a^*$  too much requires that, in expectation, each of the samples inducing the estimate  $\hat{Q}(s_0(H), a^*)$  does not underestimate too much the true value of  $a^*$ . This implies that all of the outcome states of  $a^*$  should “identify” their optimal actions, which in turn translates into accuracy requirements posed to all (both optimal and sub-optimal) actions applicable at these outcome states of  $a^*$ . As we just mentioned, the accuracy requirements from sub-optimal actions in MC are negligible, and thus the effective burden is only with the accuracy requirements from the optimal actions at the outcome states.

In sum, the requirement of not underestimating the optimal action translates to accuracy requirements from the optimal actions at the outcome states, *at different stages of planning*. In the lack of more effective proof methods, the accuracy of each estimation sample in the proof of Proposition 2 is considered in isolation, and this is precisely the point where the difference between the bound and the actual performance may inflate. Indeed, the accuracy of an action estimate correlates with the accuracy of the same action estimate at subsequent points in time, yet this correlation is not factored into the bounds.

Importantly, the analysis of  $MC(\alpha)$  in that respect is not any different: partial averaging does not offer a way to factor this correlation, but only reduces the bound by imposing accuracy requirements on fewer samples. Clearly, as the number of iterations increases, the correlation increases as well. The question, however, is when can we expect to have higher correlation at earlier stages. For instance, when the probability mass of the transition function concentrates on a small number of outcomes, the effective action branching becomes smaller than the nominal action branching  $B$ . In such cases, one should expect to have more samples based on overlapping rollouts, and thus to have a higher correlation. In any case, when the correlation is high, MC becomes equivalent to DP in terms of the accuracy requirement on the optimal action.

In summary, the dependency of DP on  $B$  and  $K$  is of order  $(BK)^H$ , whereas, depending on the correlation, the dependency of MC on  $B$  and  $K$  can be of order as small as  $B^H$ . Therefore, MC can be expected to be less sensitive than DP to  $K$ , especially when the effective action branching is relatively low, and thus the correlation is relatively high.

**The shape of the reward function** If right from the first steps, the immediate rewards of the optimal actions appear more attractive than these of the suboptimal actions, then identifying the optimal action

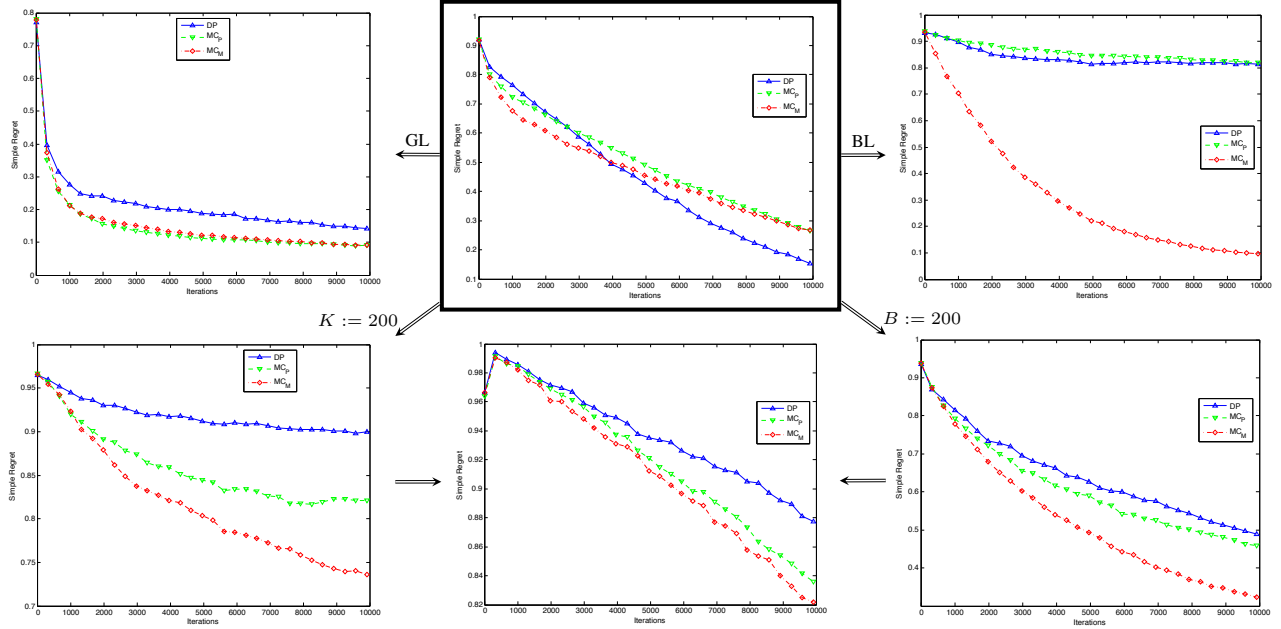
$a^*$  at  $s_0(H)$  is somewhat a simpler problem. The more challenging cases in that respect are when the discriminative rewards are pushed down the search tree, similarly to what happens in goal-driven MDPs. In such cases, identifying the optimal action  $a^*$  at  $s_0(H)$  requires properly identifying optimal actions far from the root, where samples are much sparser. Relating this point to the previous discussion on the size of the problem, it can be expected that the advantage of MC in the more challenging cases becomes more dependent on the effective action branching being relatively low.

**The entropy of the transition function** For all the bounds, the factor  $\frac{1}{B}^H$  in the exponent results from the worst-case transition probability function, which, for each action, induces a uniform distribution over its outcomes. Clearly, as the entropy of the transition functions decreases, the better the bounds and performance of both DP and MC would be. However, here as well, some differences are expected depending on the update scheme. Since DP and  $MC_P$  use in their UPDATE the estimated transition function, their value estimations would be skewed towards the value of the more probable outcomes. Although this skew decreases with the number of samples, this decrease is slower at the deeper nodes since they are sampled less frequently.  $MC_M$ , on the other hand, is free from this type of inaccuracy and therefore has certain advantage in that respect.

## 4 EXPERIMENTAL STUDY

In what follows, we put the qualitative comparison above into an empirical test. In previous work, the empirical effectiveness of online MDP planning algorithms was typically examined on a set of specific MDP problems, such as the benchmark suites of planning competitions (IPPC). These benchmarks, however, are problematic to use if one wants to examine the marginal impact of various parameters of the MDPs on the effectiveness of the algorithms, because these parameters simply cannot be controlled. In fact, almost all of these benchmarks are too large to compute the actual value of different actions at a state, and without that, assessing simple regret of different algorithms is impossible. Taking that on board, we devised a parametric MDP model from which one can select MDP instances with (i) arbitrary set of action values at the initial state, and (ii) arbitrary setting of the parameters discussed in the previous section. This allows us to experiment with large MDPs, for which otherwise it would be impossible (in reasonable time and computational resources) to compute the value function, and based on it, assess simple regret of different algorithms.

For ease of presentation, in what follows we refer to nodes  $s(h)$  simply by  $s$ ; the steps-to-go component of the nodes remains clear from the text. In our base MDP setup, the horizon is set to  $H = 10$ , there are exactly  $K = 20$  actions applicable at every node, and each node/action pair induces exactly  $B$  equiprobable outcome nodes, exclusive to that node/action pair, i.e. the induced state-space is tree-structured, and the transition probability functions are all uniform. For a node  $s$ , an applicable action  $a$ , and a possible outcome  $s'$ , the immediate reward is set to  $R(s, a, s') = \frac{Q(s, a)}{B}$ , and the value of the outcome node receives the remainder  $V(s') = Q(s, a) - R(s, a, s')$ . At any node  $s$ , there is exactly one optimal action, the value of which equals the value  $V(s)$  of the node, whereas all other actions have identical values of  $\epsilon V(s)$ , for some  $\epsilon \in (0, 1)$ . The choice of  $\epsilon$  plays an important role here. If, for instance,  $\epsilon$  is set equally for all nodes, then basing the value updates in both MC and DP on random (and not empirically best) action successors will surface the optimal action at  $s_0$ , and this because all the actions will be underestimated by



**Figure 2.** Experimental results on the base setup with  $K = 20$ ,  $B = 20$ , and all action outcomes being equiprobable (top-center), as well as on the variants with ( $\swarrow$ )  $K = 200$ , ( $\searrow$ )  $B = 200$ , ( $\downarrow$ )  $K = 200$  and  $B = 200$ , ( $\leftarrow$ ) “good likely” transition functions, and ( $\rightarrow$ ) “bad likely” transition functions.

a similar magnitude. Therefore, in our setup, for all nodes reachable by the optimal policy from  $s_0$ , we set  $\epsilon = 0.6$ , while for all nodes off the optimal policy, we set  $\epsilon = 0.8$ . The only node that is not properly covered by this categorization is the actual initial node  $s_0$ , and there we also set  $\epsilon = 0.8$ . In this setup, the estimates induced by random updates would not preserve the right order of the action values, imposing a harder challenge on the algorithms.

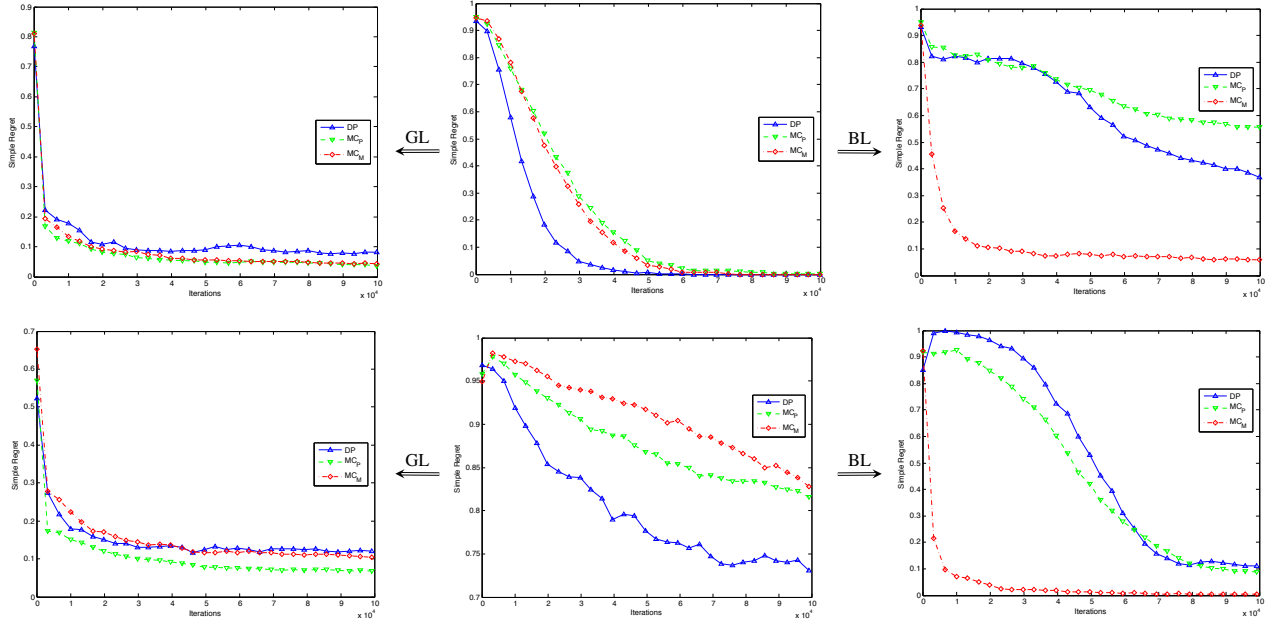
The emphasized plot in the top-center of Figure 2 depicts the simple regret obtained by the three examined algorithms, DP,  $MC_P$ , and  $MC_M$ , on the base setup as above, as a function of the number of iterations. While MC here appear slightly better at the start, DP quickly catches up and gradually outperforms MC. However, the picture changes when the base setup is modified in several different ways. First, when scaling up the problem by increasing  $K$  to 200 (bottom-left), or by increasing  $B$  to 200 (bottom-right), or both (bottom-center), MC performs much better than DP at all times, with  $MC_M$  being the clear dominator. Second, the top-left and top-right plots in Figure 2 depict the results for two setups that deviate from the base only by altering the entropy of the transition probability functions. In both setups, for each node  $s$  and each applicable action  $a$ , one outcome  $s'$  is substantially more likely than all other, with  $\mathbb{P}(s' | s, a) = 0.9$  and, for all outcomes  $s'' \neq s'$ ,  $\mathbb{P}(s'' | s, a) = \frac{0.1}{B-1}$ . The difference between the two setups is the relative value of the more likely outcome  $s'$ : In the “good likely” setup (GL), the more likely outcome is also more valuable, i.e.,  $R(s, a, s') + V(s') > R(s, a, s'') + V(s'')$  for all  $s'' \neq s'$ , and in the “bad likely” setup (BL), it is the other way around. In either case, all action-outcome values are set such that (1) they reflect the value of the action, i.e.  $\sum_{s'} \mathbb{P}(s' | s, a) (R(s, a, s') + V(s')) = Q(s, a)$ , and (2) each action-outcome value is neither smaller than half of the action value, nor higher than the maximal immediate reward ( $= 1$ , in our experiments), times the number of steps-to-go.

In both “good likely” and “bad likely” variants, the reduction in the entropy of the transition function basically reduces the effective state branching of the actions, and thus the correlation between the

successive samples in MC is expected to grow, getting more in line with the optimistic assumption on MC’s dependence on  $B$  and  $K$ . The results depicted in Figure 2 support this expectation. Moving from the base setup, the performance of MC improves in both “good likely” and “bad likely” setups, and in fact, in both setups, MC outperforms DP. Likewise, importantly, while in “good likely” there is effectively no difference between  $MC_M$  and  $MC_P$ , in “bad likely”,  $MC_M$  is performing much better than  $MC_P$ , with the latter meeting the very poor performance of DP. Basically, the “bad likely” setup demonstrates how dramatic can be the implications of establishing value estimation on the estimated transition probabilities. Here, the underestimation of the values by DP and  $MC_P$  results in their very poor performance. To recap Figure 2, it appears that MC outperforms DP except for on MDPs of relatively small size, and  $MC_M$  being justifiably more robust than  $MC_P$ .

Another important aspect that we examined in our experiments pertains to the dependence of the algorithm performance on the shape of the rewards as a function of the node depth. In the base setup, at any node, the actions are rewarded proportionally to their actual value, and thus, in particular, optimal actions have higher immediate rewards than the sub-optimal actions.

Figure 3 shows the results for two setups that deviate from the base setup in that aspect as follows. (Both these setups are more challenging than the base, and thus the x-axis in Figure 3 goes up to  $10^5$  iterations, and not to  $10^4$  iterations like in Figure 2.) In “first equal” (top-center), the immediate rewards differ from the base setup only at the root, where, instead of rewarding the optimal action higher than the sub-optimal actions, all the actions have the same reward of 0.5, independently of the outcome. The results for the “good likely” and “bad likely” variants of “first equal” are depicted in top-left and top-right corners of Figure 3. In the even more challenging setup “first few equal” (bottom-center), the immediate rewards are set to the minimum between 0.5 and the action-outcome value, that is  $R(s, a, s') = \min\{0.5, Q(s, a)\}$ ; in the “good likely” (bottom-left) and “bad likely” (bottom-right) variants, the appropriate factor



**Figure 3.** Experimental results on the “first equal” (top-center) and “first few equal” (bottom-center) modifications of the base setup, as well as on their variants with “good likely” (←) and “bad likely” (→) transition functions.

is added to  $Q(s, a)$ .

Comparing the results for the variants of the base setup in Figure 2 with the results for “first equal” and “first few equal” in Figure 3, the qualitative relative performance of DP,  $MC_P$ , and  $MC_M$  remains the same, with the absolute performance of all algorithm decreasing, as expected, from the base setup to “first equal”, and from “first equal” to “first few equal”. It should also be noted that here, in contrast to the base setup, the advantage of DP over MC under equiprobable action outcomes was observed also when  $K$  and  $B$  were higher than 20. This goes in line with the dependence of MC’s performance on the correlation between the successive samples, because pushing the discriminative rewards down the tree delays the correlation. Finally, we also experimented with various graph-structured (in contrast to tree-structured) variants of our MDP model. As expected, the performance of all three algorithms improved with the degree of the multi-connectedness of the nodes, but the improvements were of the same magnitude for all three algorithms. In sum, based on our experiments and in line with the analysis in Section 3, DP appear more effective than MC as long as the size of the problem is sufficiently small, but otherwise, MC outperforms DP even under most challenging conditions, especially if the probability mass of the transition functions concentrates on very few outcomes.

## ACKNOWLEDGEMENTS

This work was partially supported by the FA8655-12-1-2096, and the EOARD grant ISF grant 1045/12.

## REFERENCES

- [1] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.
- [2] R. Bjarnason, A. Fern, and P. Tadepalli, ‘Lower bounding Klondike Solitaire with Monte-Carlo planning’, in *ICAPS*, (2009).
- [3] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, ‘A survey of Monte-Carlo tree search methods’, *IEEE Trans. on Comp. Intell. and AI in Games*, 143, (2012).
- [4] S. Bubeck and R. Munos, ‘Open loop optimistic planning’, in *COLT*, pp. 477–489, (2010).
- [5] S. Bubeck, R. Munos, and G. Stoltz, ‘Pure exploration in finitely-armed and continuous-armed bandits’, *Theor. Comput. Sci.*, **412**(19), 1832–1852, (2011).
- [6] T. Cazenave, ‘Nested Monte-Carlo search’, in *IJCAI*, pp. 456–461, (2009).
- [7] P.-A. Coquelin and R. Munos, ‘Bandit algorithms for tree search’, in *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 67–74, Vancouver, BC, Canada, (2007).
- [8] P. Eyerich, T. Keller, and M. Helmert, ‘High-quality policies for the Canadian Traveler’s problem’, in *AAAI*, (2010).
- [9] Z. Feldman and C. Domshlak, ‘Simple regret optimization in online planning for Markov decision processes’, *CoRR*, arXiv:1206.3382v2 [cs.AI], (2012).
- [10] Z. Feldman and C. Domshlak, ‘Monte-Carlo planning: Theoretically fast convergence meets practical efficiency’, in *UAI*, (2013).
- [11] Z. Feldman and C. Domshlak, ‘Monte-Carlo tree search: To MC or to DP?’, Technical Report IE/IS-2014-03, Technion, (2014).
- [12] Z. Feldman and C. Domshlak, ‘On MABs and separation of concerns in Monte-Carlo planning for MDPs’, in *ICAPS*, (2014).
- [13] S. Gelly and D. Silver, ‘Monte-Carlo tree search and rapid action value estimation in computer Go’, *AIJ*, **175**(11), 1856–1875, (2011).
- [14] T. Keller and M. Helmert, ‘Trial-based heuristic tree search for finite horizon MDPs’, in *ICAPS*, pp. 135–143, (2013).
- [15] L. Kocsis and C. Szepesvári, ‘Bandit based Monte-Carlo planning’, in *ECML*, pp. 282–293, (2006).
- [16] L. Péret and F. Garcia, ‘On-line search for solving Markov decision processes via heuristic sampling’, in *ECAI*, pp. 530–534, (2004).
- [17] M. Puterman, *Markov Decision Processes*, Wiley, 1994.
- [18] H. Robbins, ‘Some aspects of the sequential design of experiments’, *Bull. Amer. Math. Soc.*, **58**(5), 527535, (1952).
- [19] C. D. Rosin, ‘Nested rollout policy adaptation for Monte Carlo tree search’, in *IJCAI*, pp. 649–654, (2011).
- [20] N. Sturtevant, ‘An analysis of UCT in multi-player games’, in *CCG*, p. 3749, (2008).
- [21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
- [22] D. Tolpin and S. E. Shimony, ‘MCTS based on simple regret’, in *AAAI*, (2012).

# On MABs and Separation of Concerns in Monte-Carlo Planning for MDPs

Zohar Feldman and Carmel Domshlak

Technion—Israel Institute of Technology

Haifa, Israel

{zoharf@tx, dcarmel@ie}.technion.ac.il

## Abstract

Linking online planning for MDPs with their special case of stochastic multi-armed bandit problems, we analyze three state-of-the-art Monte-Carlo tree search algorithms: UCT, BRUE, and MaxUCT. Using the outcome, we (i) introduce two new MCTS algorithms, MaxBRUE, which combines uniform sampling with Bellman backups, and MpaUCT, which combines UCB1 with a novel backup procedure, (ii) analyze them formally and empirically, and (iii) show how MCTS algorithms can be further stratified by an exploration control mechanism that improves their empirical performance without harming the formal guarantees.

## Introduction

In online planning for MDPs, the agent focuses on its current state only, deliberates about the set of possible policies from that state onwards and, when interrupted, chooses what action to perform next. In formal analysis of algorithms for online MDP planning, the quality of the action  $a$ , chosen for state  $s$  with  $H$  steps-to-go, is assessed in terms of the *simple regret* measure, capturing the performance loss that results from taking  $a$  and then following an optimal policy  $\pi^*$  for the remaining  $H - 1$  steps, instead of following  $\pi^*$  from the beginning (Bubeck and Munos 2010).

Most algorithms for online MDP planning constitute variants of what is called Monte-Carlo tree search (MCTS) (Sutton and Barto 1998; Péret and Garcia 2004; Kocsis and Szepesvári 2006; Coquelin and Munos 2007; Cazenave 2009; Rosin 2011; Tolpin and Shimony 2012). When the MDP is specified declaratively, that is, when all its parameters are provided explicitly, the palette of algorithmic choices is wider (Bonet and Geffner 2012; Kolobov, Mausam, and Weld 2012; Busoniu and Munos 2012; Keller and Helmert 2013). However, when only a generative model of MDP is available, that is, when the actions of the MDP are given only by their “black box” simulators, MCTS algorithms are basically the only choice. In MCTS, agent deliberation is based on simulated sequential sampling of the state space. MCTS algorithms have also

become popular in other settings of sequential decision making, including those with partial state observability and adversarial effects (Gelly and Silver 2011; Sturtevant 2008; Bjarnason, Fern, and Tadepalli 2009; Balla and Fern 2009; Eyerich, Keller, and Helmert 2010; Browne et al. 2012).

The popularity of MCTS methods is due in part to their ability to deal with generative problem representations, but they have other desirable features as well. First, while MCTS algorithms can natively exploit problem-specific heuristic functions, their correctness is independent of the heuristic’s properties, and they can as well be applied without any heuristic information whatsoever. Second, numerous MCTS algorithms exhibit strong anytime-ness: not only can a meaningful action recommendation be provided at any interruption point instantly, in time  $O(1)$ , but the quality of the recommendation also improves very smoothly, in time steps that are independent of the size of the explored state space.

Fundamental developments in the area of MCTS algorithms can all be traced back to stochastic multi-armed bandit (MAB) problems (Robbins 1952). Here we take a closer look at three state-of-the-art MCTS algorithms, UCT (Kocsis and Szepesvári 2006), BRUE (Feldman and Domshlak 2012; 2013), and MaxUCT (Keller and Helmert 2013), linking them to algorithms for online planning in MABs. This analysis leads to certain interesting realizations about the examined MCTS algorithms. Taking these realizations as our point of departure, we:

- Introduce two new MCTS algorithms, MaxBRUE, which combines uniform sampling with Bellman backups, and MpaUCT which combines UCB1 with a novel backup procedure;
- Establish formal guarantees of exponential-rate convergence for MaxBRUE (that turn out to be even stronger than those known to be provided by BRUE), and a hint about the polynomial-rate convergence of MpaUCT;
- Demonstrate empirically that, in line with the empirical analysis of pure exploration in MAB (Bubeck and Munos 2010), MaxBRUE performs better than

MpaUCT and MaxUCT under a permissive planning-time allowance, while the opposite holds under short planning times;

- Show how MaxBRUE (and probably other algorithms) can be stratified by an exploration control mechanism that substantially improves the empirical performance without harming the formal guarantees.

## Background

Henceforth, the operation of drawing a sample from a distribution  $\mathcal{D}$  over set  $\mathbb{N}$  is denoted by  $\sim \mathcal{D}[\mathbb{N}]$ ,  $\mathcal{U}$  denotes uniform distribution, and  $[n]$  for  $n \in \mathbb{N}$  denotes the set  $\{1, \dots, n\}$ . For a sequence of tuples  $\rho$ ,  $\rho[i]$  denotes the  $i$ -th tuple along  $\rho$ , and  $\rho[i].x$  denotes the value of the field  $x$  in that tuple.

**Markov Decision Processes.** MDP is a standard model for planning under uncertainty (Puterman 1994). An MDP  $\langle S, A, \mathbb{P}, R \rangle$  is defined by a set of states  $S$ , a set of state transforming actions  $A$ , a stochastic transition function  $\mathbb{P} : S \times A \times S \rightarrow [0, 1]$ , and a reward function  $R : S \times A \times S \rightarrow \mathbb{R}$ . The states are fully observable and, in the finite horizon setting considered here, the rewards are accumulated over some predefined number of steps  $H$ . In what follows,  $s\langle h \rangle$  denotes an MDP state  $s$  with  $h$  steps-to-go, and  $A(s) \subseteq A$  denotes the actions applicable in state  $s$ . The objective of planning in MDPs is to sequentially choose actions so as to maximize the accumulated reward. The representation of large-scale MDPs can be either declarative or generative, but anyway concise, and allowing for simulated execution of all feasible action sequences, from any state of the MDP. Henceforce, the state and action branching factors of the MDP in question are denoted by  $K = \max_s |A(s)|$  and  $B = \max_{s,a} |\{s' \mid \mathbb{P}(s'|s,a) > 0\}|$  respectively.

**Simple Regret Minimization in MAB.** A stochastic multi-armed bandit (MAB) problem is an MDP defined over a single state  $s$ . The actions in MABs do not affect the state, but are associated with stochastic rewards. Most research on MABs has been devoted to the setup of reinforcement learning-while-acting, where the cumulative regret is of interest and exploration must be intertwined with exploitation. For this setup, an action selection strategy called UCB1 was shown to attain the optimal logarithmic cumulative regret by balancing the empirical attractiveness of the actions with the potential of less sampled actions. Specifically, UCB1 samples each action once, and then iteratively selects actions as

$$\operatorname{argmax}_a \left[ \hat{\mu}_a + \alpha \sqrt{\frac{\log n}{n_a}} \right],$$

where  $n$  is the total number of samples so far,  $n_a$  is the number of samples that went to action  $a$ , and  $\hat{\mu}_a$  is the average reward of these samples of  $a$ . The parameter  $\alpha$  is an exploration factor that balances the two components of the UCB1 formula.

In contrast to learning-while-acting, in online planning for MAB the agent is provided with a simula-

	EBA	MPA
uniform	$\bigcirc e^{-\bigcirc n}$	—
UCB( $\alpha$ )	$\bigcirc n^{-\bigcirc}$	$\bigcirc n^{-2(\alpha-1)}$

Table 1: Upper bounds on the expected simple regret of some online planning algorithms for MAB (Bubeck and Munos 2010)

tor that can be used “free of charge” to evaluate the alternative actions by drawing samples from their reward distributions. An algorithm for online planning for MAB is defined by an exploration strategy, used to sample the actions, and a recommendation strategy, used at the end of the planning to select an action that is believed to minimize simple regret. Recently, Bubeck et al. (2010) investigated worst-case convergence-rate guarantees provided by various MAB planning algorithms; their key findings are depicted in Table 1. Two exploration strategies, the uniform one and a generalization of UCB1, have been examined in the context of “the empirical best action” (EBA) and “the most played action” (MPA) recommendation strategies. The table provides upper bounds on the expected simple regret of the considered pairs of exploration (rows) and recommendation (columns) strategies, whereas the  $\bigcirc$  symbols are distribution-dependent constants. Bubeck et al. (2010) also examined these algorithms empirically, and showed that, in line with the details of their formal analysis, the UCB1-based exploration strategy outperforms uniform+EBA under a moderate number of samples, while the opposite holds under more permissive exploration budgets.

**Monte-Carlo Tree Search.** MCTS, a canonical scheme underlying various MCTS algorithms for online MDP planning, is depicted in Figure 1. MCTS explores the state space in the radius of  $H$  steps from the initial state  $s_0$  by iteratively issuing simulated rollouts from  $s_0$ . Each such rollout  $\rho$  comprises a sequence of simulated steps  $\langle s, a, r, s' \rangle$ , where  $s$  is a state,  $a$  is an action applicable in  $s$ ,  $r$  is an immediate reward collected from issuing the action  $a$ , and  $s'$  is the resulting state. In particular,  $\rho[0].s = s_0$  and  $\rho[t].s' = \rho[t+1].s$  for all  $t$ .

Each generated rollout is used to update some variables of interest. These variables typically include at least the action value estimators  $\hat{Q}(s\langle h \rangle, a)$ , as well as the counters  $n(s\langle h \rangle, a)$  that record the number of times the corresponding estimators  $\hat{Q}(s\langle h \rangle, a)$  have been updated. Instances of MCTS vary mostly along the different implementation of the strategies STOP-ROLLOUT, specifying when to stop a rollout; SELECT-ACTION, prescribing the action to apply in the current state of the rollout; and UPDATE, specifying how a rollout should update the maintained variables.

Once interrupted, MCTS uses the information collected throughout the exploration to recommend an action to perform at state  $s_0$ . The rollout-based exploration of MCTS is especially appealing in the setup of online planning because it allows smooth improvement

**MCTS:** [input:  $\langle S, A, \mathbb{P}, R \rangle$ ;  $s_0 \in S$ ]

**while** time permits **do**

$\rho \leftarrow \text{ROLLOUT}$     *// generate rollout*  
 $\text{UPDATE}(\rho)$

**return**  $\arg \max_a \widehat{Q}(s_0 \langle H \rangle, a)$

**procedure** ROLLOUT

$\rho \leftarrow \langle \rangle$ ;  $s \leftarrow s_0$ ;  $t \leftarrow 0$

**while not** STOP-ROLLOUT( $\rho$ ) **do**

$a \leftarrow \text{SELECT-ACTION}(s, t)$

$s' \leftarrow \text{SAMPLE-OUTCOME}(s, a, t)$

$r \leftarrow R(s, a, s')$

$\rho[t] \leftarrow \langle s, a, r, s' \rangle$

$s \leftarrow s'$ ;  $t \leftarrow t + 1$

**return**  $\rho$

Figure 1: Monte-Carlo tree search

of the intermediate quality of recommendation by propagating to the root information from states at deeper levels in iterations of low complexity of  $O(H)$ .

**MCTS algorithms: UCT, BRUE, and MaxUCT.**

UCT, one of the most popular algorithms for online MDP planning to date, is depicted in Figure 2 as a particular instantiation of MCTS. In UCT, the rollouts end at terminal states, i.e., at depth  $H$  or at states with no applicable actions.<sup>1</sup> Each rollout updates all value estimators  $\widehat{Q}(s \langle h \rangle, a)$  of the  $(s \langle h \rangle, a)$  pairs encountered along the rollout. The estimators are updated via the MC-BACKUP procedure, which averages the accumulated reward of the rollouts from  $s \langle h \rangle$  to terminal states.

Under this flow, a necessary condition for the value estimators to converge to their true values is that the portion of samples that correspond to selections of optimal actions must tend to 1 as the number of samples increases. At the same time, in order to increase the confidence that the optimal actions will be recognized at the nodes, all the applicable actions must be sampled infinitely often. The sampling strategy of UCT, UCB1, aims at achieving precisely that: UCB1 ensures that each action is selected at least a logarithmic number of times, and that suboptimal actions are selected at most a logarithmic number of times; thus, the proportion of best-action selections indeed tends to 1.

UCT has many success stories and much of this success is accounted for by the exploitative property of UCT. This property results in skewing towards more attractive actions right from the beginning of exploration, a protocol that presumably enables fast homing on “good” actions. However, Table 1 shows that exploitation may considerably slow down the reduction of simple regret over time. Indeed, much like UCB1 for MABs, UCT achieves only polynomial-rate reduc-

<sup>1</sup>In a more popular version of UCT, a rollout ends at a newly encountered node, but this is secondary to our discussion.

**procedure** UPDATE( $\rho$ )

$\bar{r} \leftarrow 0$

**for**  $d \leftarrow |\rho|, \dots, 1$  **do**

$h \leftarrow H - d$

$a \leftarrow \rho[d].a$

$n(s \langle h \rangle) \leftarrow n(s \langle h \rangle) + 1$

$n(s \langle h \rangle, a) \leftarrow n(s \langle h \rangle, a) + 1$

$\bar{r} \leftarrow \bar{r} + \rho[d].r$

MC-BACKUP( $s \langle h \rangle, a, \bar{r}$ )

**procedure** MC-BACKUP( $s \langle h \rangle, a, \bar{r}$ )

$\widehat{Q}(s \langle h \rangle, a) \leftarrow \frac{n(s \langle h \rangle, a) - 1}{n(s \langle h \rangle, a)} \widehat{Q}(s \langle h \rangle, a) + \frac{1}{n(s \langle h \rangle, a)} \bar{r}$

**procedure** STOP-ROLLOUT( $\rho$ )

$t \leftarrow |\rho|$

**return**  $t = H$  **or**  $A(\rho[t].s') = \emptyset$

**procedure** SELECT-ACTION( $s, d$ )    *// UCB*

$h \leftarrow H - d$

**if**  $\exists a : n(s \langle h \rangle, a) = 0$  **then**

**return**  $a$

**return**  $\arg \max_a \left[ \widehat{Q}(s \langle h \rangle, a) + c \sqrt{\frac{\log n(s \langle h \rangle)}{n(s \langle h \rangle, a)}} \right]$

**procedure** SAMPLE-OUTCOME( $s, a, t$ )

**return**  $s' \sim \mathbb{P}(S | s, a)$

Figure 2: UCT algorithm as a specific set of sub-routines for MCTS

tion of simple regret over time (Bubeck, Munos, and Stoltz 2011), and the number of samples after which the bounds of UCT on simple regret become meaningful might be as high as hyper-exponential in  $H$  (Coquelin and Munos 2007).

Using this observation and following the findings of Bubeck et al. (Bubeck and Munos 2010), in our earlier work we introduced the concept of “separation of concerns,” whereby the first part of each rollout is devoted solely to the purpose of selecting particular nodes, whereas the second part is devoted to estimating their value (Feldman and Domshlak 2012). We showed a specific algorithm, BRUE, that implements this concept by always updating value estimators with samples that activate currently best actions only, but the estimators to be updated are chosen by rolling out actions uniformly at random. It turns out that, in contrast to UCT, BRUE achieves an exponential-rate reduction of simple regret over time, with the bounds on simple regret becoming meaningful after only exponential in  $H^2$  number of samples. Moreover, BRUE was also shown to be very effective in practice.

Finally, BRUE was not the only successful attempt to improve over UCT. In particular, Keller & Helmert (2013) recently introduced MaxUCT, a modification of UCT in which MC backups are replaced with Bellman backups using approximate transition probabilities, and demonstrated that MaxUCT substantially outperforms



	EBA	MPA	UCB1
Uniform	BRUE, <u>MaxBRUE</u>	—	—
UCB1	MaxUCT	<u>MpaUCT</u>	UCT

Table 2: MCTS algorithms for MDPs through the lens of the MAB exploration topology. Rows are exploration strategies, and columns are recommendation strategies.

UCT empirically. In terms of formal guarantees, however, there is no dramatic difference between the convergence rates of the two algorithms.

### From MAB to MDP

Relating between online planning for MAB and for more general MDPs, we begin by drawing ties between

- (i) MCTS rollout sampling strategies and arm exploration strategies in MAB, and
- (ii) MCTS selection of actions used to update search nodes and arm recommendation strategies in MAB.

Considering the three algorithms discussed above in that perspective, the picture appears to be as follows.

- UCT combines MC backups with a rollout sampling driven by the UCB1 action-selection strategy. Interestingly, there is no perfect analogy between UCT and a reasonable algorithm for pure exploration in MAB. This is because, at all nodes but the root, UCB1 *de facto* drives *both* UCT’s rollout sampling and node updates, yet recommending an arm in MAB according to UCB1 does not have well justified semantics.
- BRUE is analogous to uniform exploration with empirical best action recommendation: Applying the principle of separation of concerns, nodes are reached by selecting actions uniformly, and the samples used in the MC backups are generated by selecting the empirical best actions.
- MaxUCT is analogous to UCB( $\alpha$ ) exploration with best empirical action recommendation: With Bellman backups, the updated value corresponds to the value of the action with the best empirical value. Interestingly, this perspective reveals that switching from MC backups to Bellman backups in MaxUCT essentially constitutes another way to separate concerns in the sense discussed above.

Building on this link between online planning for MAB and general MDPs, in what follows we present and analyze two new MCTS algorithms for MDP planning. The union of the known and new algorithms is depicted in Table 2, with the names of the new algorithms underscored.

The first algorithm, MpaUCT, is analogous to UCB( $\alpha$ ) exploration with “most played action” recommendations in MAB. This algorithm is inspired by the

```

procedure UPDATE( $\rho$ )
  for  $d \leftarrow |\rho|, \dots, 1$  do
     $h \leftarrow H - d$ 
     $a \leftarrow \rho[d].a$ 
     $s' \leftarrow \rho[d].s'$ 
     $n(s\langle h \rangle) \leftarrow n(s\langle h \rangle) + 1$ 
     $n(s\langle h \rangle, a) \leftarrow n(s\langle h \rangle, a) + 1$ 
     $n(s\langle h \rangle, a, s') \leftarrow n(s\langle h \rangle, a, s') + 1$ 
     $\hat{R}(s\langle h \rangle, a) = \hat{R}(s\langle h \rangle, a) + \rho[d].r$ 
    MPA-BACKUP( $s\langle h \rangle, a$ )

```

```

procedure MPA-BACKUP( $s\langle h \rangle, a$ )
   $\hat{Q}(s\langle h \rangle, a) \leftarrow \frac{\hat{R}(s\langle h \rangle, a)}{n(s\langle h \rangle, a)}$ 
   $v \leftarrow 0$ 
  for  $s' \in \{s' \mid n(s\langle h \rangle, a, s') > 0\}$  do
     $A \leftarrow \operatorname{argmax}_{a'} n(s'\langle h-1 \rangle, a')$ 
     $a^* \leftarrow \operatorname{argmax}_{a' \in A} \hat{Q}(s'\langle h-1 \rangle, a')$ 
     $v \leftarrow v + \frac{n(s\langle h \rangle, a, s')}{n(s\langle h \rangle, a)} \hat{Q}(s'\langle h-1 \rangle, a^*)$ 
   $\hat{Q}(s\langle h \rangle, a) \leftarrow \hat{Q}(s\langle h \rangle, a) + v$ 

```

Figure 3: MpaUCT as UCT with a modified UPDATE procedure

findings of Bubeck et al. (Bubeck, Munos, and Stoltz 2011) that, unlike all other bounds shown in Table 1, the convergence rate of UCB( $\alpha$ )+MPA planning on MAB can be bounded independently of the problem parameters.

A simple adaptation of the MPA recommendation strategy to MDPs is a modification of the Bellman backup: Instead of folding up the value of the empirically best action, we propagate the value of the action that was updated the most. Ties are broken in favor of actions with better empirical value. The resulting algorithm is depicted in Figure 3, and later on we present our empirical findings with it.

The second algorithm, MaxBRUE, is—like BRUE— analogous to uniform exploration with EBA recommendations, but it employs Bellman backups rather than MC backups. MaxBRUE is depicted in Figure 4. As we show in the proof of Theorem 1 below, not only does MaxBRUE achieve exponential-rate reduction of simple regret similarly to BRUE, but the particular parameters of the convergence bounds are more attractive than those currently known for BRUE. Basically, Theorem 1 positions MaxBRUE as the worst-case most efficient MCTS algorithm for online MDP planning to date.

**Theorem 1** *Let MaxBRUE be called on a state  $s_0$  of an MDP  $\langle S, A, \mathbb{P}, R \rangle$  with rewards in  $[0, 1]$ , and finite horizon  $H$ . After  $n \geq 1$  iterations of MaxBRUE, we have the probability  $p_{\text{err}}$  of sub-optimal action choice being bounded as  $p_{\text{err}} \leq \alpha e^{-\beta n}$ , and the expected simple regret  $\Delta$  being bounded as  $\Delta \leq H\alpha e^{-\beta n}$ , where  $\alpha = 3K(3BK)^H$ ,  $\beta = \frac{\varepsilon^2}{4K(4BK)^H H^2}$ , and  $\varepsilon$  is the simple regret of the second-best action at  $s_0\langle H \rangle$ .*

**procedure** UPDATE( $\rho$ )  
**for**  $d \leftarrow |\rho|, \dots, 1$  **do**  
 $h \leftarrow H - d$   
 $a \leftarrow \rho[d].a$   
 $s' \leftarrow \rho[d].s'$   
 $n(s\langle h \rangle) \leftarrow n(s\langle h \rangle) + 1$   
 $n(s\langle h \rangle, a) \leftarrow n(s\langle h \rangle, a) + 1$   
 $n(s\langle h \rangle, a, s') \leftarrow n(s\langle h \rangle, a, s') + 1$   
 $\widehat{R}(s\langle h \rangle, a) = \widehat{R}(s\langle h \rangle, a) + \rho[d].r$   
 BELLMAN-BACKUP( $s\langle h \rangle, a$ )

**procedure** BELLMAN-BACKUP( $s\langle h \rangle, a$ )  
 $\widehat{Q}(s\langle h \rangle, a) \leftarrow \frac{\widehat{R}(s\langle h \rangle, a)}{n(s\langle h \rangle, a)}$   
 $v \leftarrow 0$   
**for**  $s' \in \{s' \mid n(s\langle h \rangle, a, s') > 0\}$  **do**  
 $v \leftarrow v + \frac{n(s\langle h \rangle, a, s')}{n(s\langle h \rangle, a)} \widehat{Q}(s'\langle h-1 \rangle, a')$   
 $\widehat{Q}(s\langle h \rangle, a) \leftarrow \widehat{Q}(s\langle h \rangle, a) + v$

**procedure** STOP-ROLLOUT( $\rho$ )  
 $t \leftarrow |\rho|$   
**return**  $t = H$  **or**  $A(\rho[t].s') = \emptyset$

**procedure** SELECT-ACTION( $s, t$ ) // uniform  
**return**  $a \sim \mathcal{U}[A(s)]$

**procedure** SAMPLE-OUTCOME( $s, a, t$ )  
**return**  $s' \sim \mathbb{P}(S \mid s, a)$

Figure 4: MaxBRUE algorithm as a specific set of sub-routines for MCTS

**Proof:** A key sub-claim we prove first is that, at any iteration of the algorithm, for all  $h \in \llbracket H \rrbracket$ , all states  $s$  reachable from  $s_0$  in  $H - h$  steps, all actions  $a \in A(s)$ , and any  $\delta > 0$ , it holds that

$$\mathbb{P} \left\{ \left| \widehat{Q}(s\langle h \rangle, a) - Q(s\langle h \rangle, a) \right| \geq \delta \right\} \leq 2(3KB)^h e^{-\frac{2\delta^2 n(s\langle h \rangle, a)}{(4BK)^h h^2}}. \quad (1)$$

The proof of this sub-claim is by induction on  $h$ . Starting with  $h = 1$ , by Hoeffding concentration inequality, we have that

$$\mathbb{P} \left\{ \left| \widehat{Q}(s\langle 1 \rangle, a) - Q(s\langle 1 \rangle, a) \right| \geq \delta \right\} \leq 2e^{-2\delta^2 n(s\langle 1 \rangle, a)}.$$

Now, assuming Eq. 1 holds for  $h' \leq h$ , we prove it holds for  $h + 1$ . From the induction hypothesis, we have

$$\begin{aligned} & \mathbb{P} \left\{ \left| \widehat{Q}(s\langle h \rangle, a) - Q(s\langle h \rangle, a) \right| \geq \delta \right\} \\ & \leq \mathbb{P} \left\{ n(s\langle h \rangle, a) \leq \frac{n(s\langle h \rangle)}{2K} \right\} + \\ & \quad \mathbb{P} \left\{ \left| \widehat{Q}(s\langle h \rangle, a) - Q(s\langle h \rangle, a) \right| \geq \delta \mid n(s\langle h \rangle, a) > \frac{n(s\langle h \rangle)}{2K} \right\} \\ & \leq e^{-\frac{n(s\langle h \rangle)}{2K^2}} + 2(3BK)^h e^{-\frac{2\delta^2 n(s\langle h \rangle)}{2K(4BK)^h h^2}} \\ & \leq \left( 1 + 2(3BK)^h \right) e^{-\frac{\delta^2 n(s\langle h \rangle)}{K(4BK)^h h^2}}, \end{aligned}$$

which implies

$$\begin{aligned} & \mathbb{P} \left\{ \left| \max_a \widehat{Q}(s\langle h \rangle, a) - Q(s\langle h \rangle, \pi^*(s)) \right| \geq \delta \right\} \\ & \leq K \left( 1 + 2(3BK)^h \right) e^{-\frac{\delta^2 n(s\langle h \rangle)}{K(4BK)^h h^2}}. \end{aligned} \quad (2)$$

Denoting  $\widehat{p}_{s,a,s'} = \frac{n(s\langle h+1 \rangle, a, s')}{n(s\langle h+1 \rangle, a)}$  and  $\widehat{p}_{s,a,s'}^{\text{diff}} = \widehat{p}_{s,a,s'} - \mathbb{P}(s' \mid s, a)$ , it holds that

$$\begin{aligned} & \left| \widehat{Q}(s\langle h+1 \rangle, a) - Q(s\langle h+1 \rangle, a) \right| \\ & = \left| \sum_{s'} \widehat{p}_{s,a,s'} \left[ R(s, a, s') + \max_{a'} \widehat{Q}(s'\langle h \rangle, a') \right] \right. \\ & \quad \left. - \sum_{s'} \mathbb{P}(s' \mid s, a) \left[ R(s, a, s') + Q(s'\langle h \rangle, \pi^*(s')) \right] \right| \\ & \leq \left| \sum_{s'} (\widehat{p}_{s,a,s'} - \mathbb{P}(s' \mid s, a)) \left[ R(s, a, s') + Q(s'\langle h \rangle, \pi^*(s')) \right] \right| \\ & \quad + \left| \sum_{s'} \widehat{p}_{s,a,s'} \left| \max_{a'} \widehat{Q}(s'\langle h \rangle, a') - Q(s'\langle h \rangle, \pi^*(s')) \right| \right|, \end{aligned}$$

and thus

$$\begin{aligned} & \mathbb{P} \left\{ \left| \widehat{Q}(s\langle h+1 \rangle, a) - Q(s\langle h+1 \rangle, a) \right| \geq \delta \right\} \\ & \leq \mathbb{P} \left\{ \left| \sum_{s'} \widehat{p}_{s,a,s'}^{\text{diff}} \left[ R(s, a, s') + Q(s'\langle h \rangle, \pi^*(s')) \right] \right| \geq \frac{\delta}{2} \right\} \\ & \quad + \mathbb{P} \left\{ \sum_{s'} \widehat{p}_{s,a,s'} \left| \max_{a'} \widehat{Q}(s'\langle h \rangle, a') - Q(s'\langle h \rangle, \pi^*(s')) \right| \geq \frac{\delta}{2} \right\} \\ & \leq 2e^{-\frac{\delta^2 n(s\langle h+1 \rangle, a)}{2(h+1)^2}} \\ & \quad + \sum_{s'} \mathbb{P} \left\{ \left| \max_{a'} \widehat{Q}(s'\langle h \rangle, a') - Q(s'\langle h \rangle, \pi^*(s')) \right| \geq \frac{\delta}{2\sqrt{B\widehat{p}_{s,a,s'}}} \right\}. \end{aligned} \quad (3)$$

In the last bounding of Eq. 3, the first term is due to Hoeffding and the second term is justified by noting that the solution to the problem

$$\underset{p}{\text{maximize}} \quad \sum_{i=1}^B \sqrt{cp_i} \quad \text{subject to} \quad \sum_{i=1}^B p_i = 1$$



is  $p = (\frac{1}{B}, \dots, \frac{1}{B})$ , with value  $\sum_{i=1}^B \sqrt{\frac{c}{B}} = \sqrt{cB}$ .  
From Eq. 2 we have

$$\begin{aligned} & \sum_{s'} \mathbb{P} \left\{ \left| \max_{a'} \widehat{Q}(s' \langle h \rangle, a') - Q(s' \langle h \rangle, \pi^*(s')) \right| \geq \frac{\delta}{2\sqrt{B\widehat{p}_{s,a,s'}}} \right\} \\ & \leq \sum_{s'} K \left( 1 + 2(3BK)^h \right) e^{-\frac{\delta^2 n(s' \langle h \rangle)}{(4BK)^{h+1} h^2 \widehat{p}_{s,a,s'}}} \\ & \leq BK \left( 1 + 2(3BK)^h \right) e^{-\frac{\delta^2 n(s \langle h+1 \rangle, a)}{(4BK)^{h+1} h^2}}. \end{aligned} \quad (4)$$

Returning now to Eq. 3, we have

$$\begin{aligned} & \mathbb{P} \left\{ \left| \widehat{Q}(s \langle h+1 \rangle, a) - Q(s \langle h+1 \rangle, a) \right| \geq \delta \right\} \\ & \leq 2e^{-\frac{\delta^2 n(s \langle h+1 \rangle, a)}{2(h+1)^2}} + BK \left( 1 + 2(3BK)^h \right) e^{-\frac{\delta^2 n(s \langle h+1 \rangle, a)}{(4BK)^{h+1} h^2}} \\ & \leq (3BK)^{h+1} e^{-\frac{\delta^2 n(s \langle h+1 \rangle, a)}{(4BK)^{h+1} (h+1)^2}}, \end{aligned} \quad (5)$$

finalizing the proof of the induction step. Now, given the sub-claim around Eq. 1 and denoting  $Q_a^{\text{diff}} = |\widehat{Q}(s_0 \langle H \rangle, a) - Q(s_0 \langle H \rangle, a)|$ , the first part of Theorem 1 is concluded by

$$\begin{aligned} p_{\text{err}} & \leq \sum_a \mathbb{P} \left\{ Q_a^{\text{diff}} \geq \frac{\varepsilon}{2} \right\} \\ & \leq \sum_a \left[ \mathbb{P} \left\{ n(s_0 \langle H \rangle, a) \leq \frac{n(s_0 \langle H \rangle)}{2K} \right\} + \right. \\ & \quad \left. \mathbb{P} \left\{ Q_a^{\text{diff}} \geq \frac{\varepsilon}{2} \mid n(s_0 \langle H \rangle, a) > \frac{n}{2K} \right\} \right] \quad (6) \\ & \leq \sum_a \left[ e^{-\frac{n}{2K^2}} + 2(3BK)^H e^{-\frac{\varepsilon^2 n}{4K(4BK)^H H^2}} \right] \\ & \leq 3K(3BK)^H e^{-\frac{\varepsilon^2 n}{4K(4BK)^H H^2}} \end{aligned}$$

The second part of Theorem 1 is obtained from the fact that the maximal loss from choosing a sub-optimal action at  $s_0 \langle H \rangle$  is  $H$ .  $\blacksquare$

### MaxBRUE+: Controlled Exploration

At any non-terminal node  $s \langle h \rangle$ , the inaccuracy of the  $Q$ -value estimators in Bellman-based MaxUCT and MaxBRUE stems from two sources: The inaccuracy in the estimation of the action transition probabilities, and the inaccuracy of the value estimators that are folded up to  $s \langle h \rangle$  from its immediate successors. The former is reduced with sampling actions at  $s \langle h \rangle$ , and the latter is reduced with sampling  $s \langle h \rangle$ 's successors.

If we wish to optimize the formal guarantees on the convergence rate, it is ideal to have these two sources of inaccuracy balanced. This can be achieved by equating the number of samples of the node with the number of samples of its immediate successors. In fact, this is precisely what is done by the seminal sparse sampling

**procedure** STOP-ROLLOUT( $\rho$ )

```

 $d \leftarrow |\rho|$ 
 $h \leftarrow H - d$ 
 $s \leftarrow \rho[d].s$ 
 $a \leftarrow \rho[d].a$ 
 $s' \leftarrow \rho[d].s'$ 
 $S(s, a) \leftarrow \{s' \mid n(s \langle h \rangle, a, s') > 0\}$ 
if  $n(s' \langle h-1 \rangle) > K \cdot |S(s, a)| \cdot n(s \langle h \rangle, a, s')$  then
  return true
return ( $d = H$  or  $A(\rho[d].s') = \emptyset$ )

```

Figure 5: MaxBRUE+ as MaxBRUE with a modified STOP-PROBE procedure

algorithm of Kerns et al. (2002) for PAC (probably approximately correct) MDP planning. However, the PAC setting does not require smooth improvement of the quality of recommendation over time, and thus the nodes can be sampled in a systematic manner. In contrast, in the online setup, smooth convergence is critical, and rollout-based exploration seems to serve this purpose quite effectively. At the same time, rollout-based exploration of MCTS leads to unbalanced node sampling.

While we might see unbalanced node sampling as an inevitable cost for a justified cause, this is not entirely so. The overall convergence rate with rollout-based exploration is dictated by the accuracy of the estimates at the nodes that are farther towards the horizon from  $s_0 \langle H \rangle$ . Due to branching, these nodes are expected to be sampled less frequently. However, since the search spaces induced by MDPs most typically form DAGs (and not just trees), a node with multiple ancestors might possibly be sampled more than each of its ancestors. In terms of the formal guarantees, the latter type of imbalance is worthless, and samples are better be diverted to nodes that are sampled less than their immediate ancestors.

A rather natural way to control this type of imbalance is by modifying the protocol for stopping a rollout. Given the last sample  $\langle s, a, r, s' \rangle$  along the ongoing rollout, if the number of samples  $n(s \langle h \rangle, a)$  is smaller than the number of samples  $n(s' \langle h-1 \rangle, a')$  of some action  $a' \in A(s')$  applicable at the resulting state  $s'$ , the rollout is stopped. Supported by the formal analysis of MaxBRUE in the proof of Theorem 1, we slightly modify the latter condition as follows:

- We replace the requirement depicted above with a weaker one whereby the overall number of updates  $n(s' \langle h-1 \rangle)$  of the resulting state  $s'$  is at least  $K$  times larger than  $n(s \langle h \rangle, a)$ .
- We multiply the counter  $n(s \langle h \rangle, a)$  by  $B \cdot \mathbb{P}(s' | a, s')$ . If  $\mathbb{P}(S|a, s')$  induces a uniform distribution over the plausible outcomes of  $a$  at  $s$ , this modification changes nothing. At the same time, for more/less probable outcomes  $s'$ , this modification implies a stronger/weaker condition, respectively.

The substitution of the sub-procedure STOP-PROBE

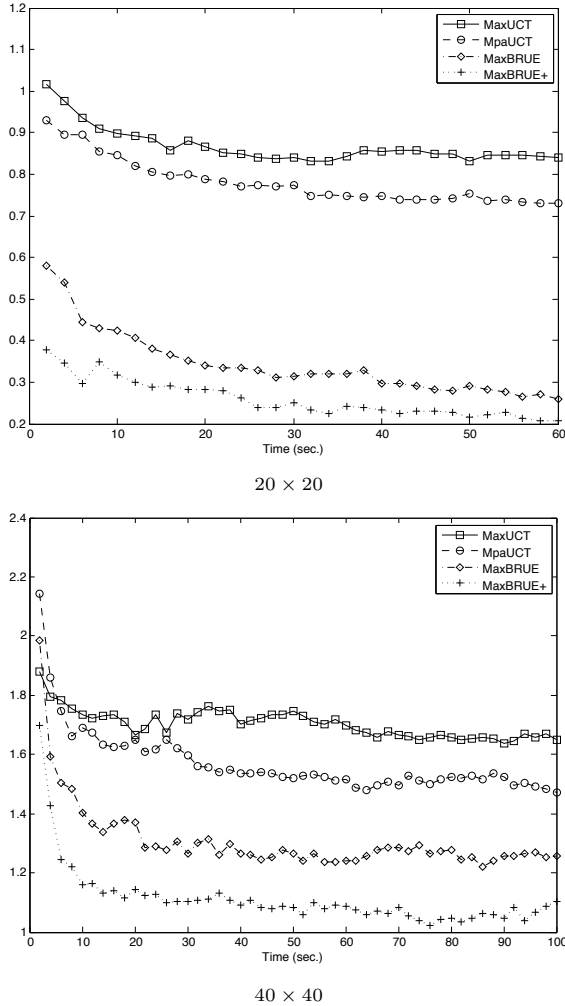


Figure 6: Simple regret reduction over time in the *Sailing* domain by different MCTS algorithms

of MaxBRUE with that depicted in Figure 5 constitutes the stratified algorithm MaxBRUE+.

Note that, in the proof of Theorem 1, we make use of the fact that  $n(s'\langle h-1 \rangle) \geq n(s\langle h \rangle, a, s')$  to replace the former with the latter in the bounding in Eq. 4. Therefore, enforcing the more conservative  $n(s'\langle h-1 \rangle) \leq K \cdot |S(s, a)| \cdot n(s\langle h \rangle, a, s')$  does not affect the bound. At the same time, it is easy to see that (i) the two sources of inaccuracy are balanced when  $n(s'\langle h-1 \rangle) = K \cdot |S(s, a)| \cdot n(s\langle h \rangle, a, s')$ , and (ii) beyond this point, the node accuracy is surpassed by the accuracy of its children.

## Experimental Evaluation

Our empirical study comprises two sets of experiments, comparing four algorithms: MaxUCT (Keller and Helmert 2013), MpaUCT, MaxBRUE, and MaxBRUE+. In the first set, we evaluate the four algorithms in terms of their reduction of simple regret in the *Sailing* do-

main (Péret and Garcia 2004). In this domain, a sailboat navigates to a goal location on an 8-connected grid, under fluctuating wind conditions. At a high level, the goal is to reach a concrete destination as quickly as possible, by choosing at each grid location a neighbor location to move to. The duration of each such move depends on the direction of the move (*ceteris paribus*, diagonal moves take  $\sqrt{2}$  more time than straight moves), the direction of the wind relative to the sailing direction (the sailboat cannot sail against the wind and moves fastest with a tail wind), and the tack. In Figure 6, we plot the empirical simple regret for increasing deliberation times for two grid sizes,  $20 \times 20$  and  $40 \times 40$ , averaged over 2000 runs with varying origin and goal locations. It is interesting to see that MaxBRUE clearly dominates MaxUCT (and MpaUCT), right from the beginning, unlike the case of BRUE and UCT whereby UCT performs better until some point.<sup>2</sup> Figure 6 also demonstrates the benefit of the exploration control of MaxBRUE+, as well as the benefit of using the MPA-backup of MpaUCT.

The second set of experiments compares between the empirical reward collected by the four algorithms on five IPPC-2011 domains, *Game-of-Life*, *SysAdmin*, *Traffic*, *Crossing*, and *Navigation*. (Almost all of the tasks in these domains are simply too large for us to examine simple regret.) From each domain, we chose four tasks,  $I_1, I_3, I_5, I_{10}$ , with higher indexes corresponding to tasks of higher complexity. Each algorithm was given a deliberation budget that decreased linearly from 10 seconds in the first step to 1 second in the last step. For each domain, Figure 7(a) depicts the general bound on the state branching factor  $K$  and the action branching factor  $B$ , as well as the specific horizon  $H$  used in the experiments, all as functions of a domain-specific parameter  $p$  that scales linearly with the instance index. ( $H > 10$  was used in goal-oriented domains.)

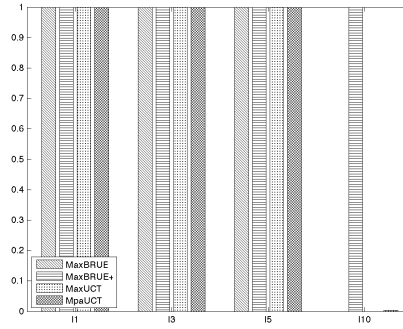
Figures 7(b-f) plot the score of the four algorithms based on 700 samples, normalized between 0 and 1 as their average improvement over a trivial baseline of random action selection. With the exception of the *SysAdmin* domain<sup>3</sup>, it appears that the results are quite similar for all algorithms. However, the relative performance differences seem to comply with the analysis of Bubeck et al. (2010) for online planning in MABs. Specifically, Bubeck et al. (2010) observed that, despite the superior convergence rate of uniform sampling in general, the bounds provided by the UCB( $\alpha$ )-based exploration can be more attractive if the sample allowance is not permissive enough with respect to the number of actions  $K$ . In case of more general MDPs, the “struc-

<sup>2</sup>For simple regret analysis on the *Sailing* domain with longer deliberation times, we refer the reader to Feldman & Domshlak (2013).

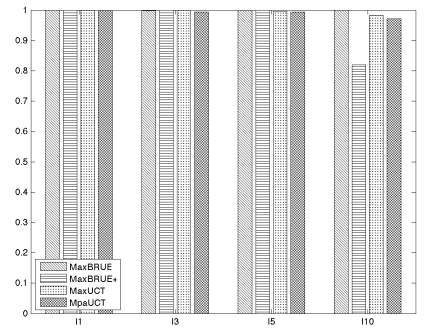
<sup>3</sup>At this stage, the dramatically superior performance of MaxBRUE+ instance  $I_{10}$  of the *Navigation* domain should be considered a positive anomaly, and not given any deep generalizing explanations.

	K	B	H
<i>Sailing</i>	8	3	$4p$
<i>Navigation</i>	5	2	40
<i>Crossing</i>	5	$2^p$	40
<i>SysAdmin</i>	$p$	$2^p$	10
<i>Game-of-Life</i>	$p^2$	$2^{p^2}$	10
<i>Traffic</i>	$2^{p^2}$	$2^{2p}$	10

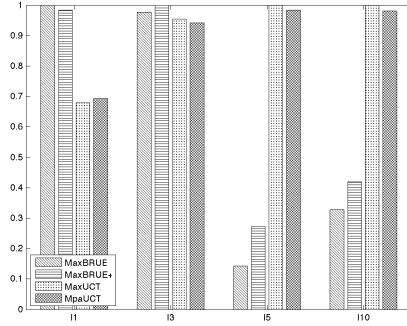
(a) Domain parameters



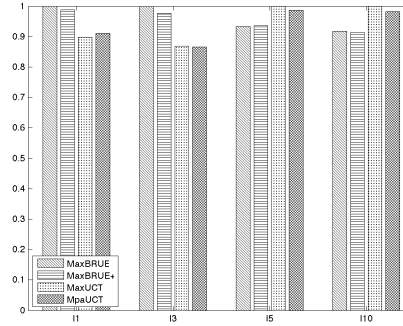
(b) Navigation



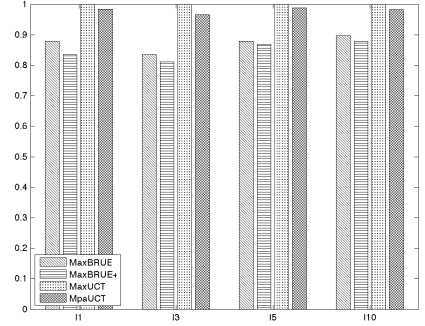
(c) Crossing



(d) SysAdmin



(e) Game-of-Life



(f) Traffic

Figure 7: (a) Structural and experimental parameters of the *Sailing* and IPPC-2011 domains, and (b-f) scores for the different MCTS algorithms as their normalized improvement over a trivial baseline

tural complexity” of the problem is determined not only by  $K$ , but also by the action branching factor  $B$  and the horizon  $H$ . In that respect, considering the “structural complexity” of the domains depicted in Figure 7(a), the results in Figures 7(b-f) are in line with the relative pros and cons of the uniform and UCB( $\alpha$ ) explorations.

- In *Sailing*, *Navigation*, and *Crossing*, both  $K$  and  $B$  grow reasonably slowly with the size of the problem. This makes our fixed time budget reasonably permissive (and thus gives the advantage to uniform exploration) across the instances.
- In the domains of intermediate structural complexity, *Game-of-Life* and *SysAdmin*, the same time budget appears to be reasonably permissive on the smaller instances, giving an advantage to uniform exploration, but then the instances grow rather fast, giving an advantage to the UCB1-based algorithms.
- In *Traffic*, both  $K$  and  $B$  grow exponentially fast with the size of the problem, making the time budget we fixed to be too small for the uniform exploration to shine even on the smallest instance.

## Summary

By drawing ties between online planning in MDPs and MABs, we have shown that the state-of-art MC planning algorithms UCT, BRUE and MaxUCT, as well as

the two newly introduced MaxBRUE and MpaUCT, borrow the theoretical and empirical properties of their MAB counterparts. In particular, we have proven that the exponential convergence of uniform exploration with recommendation of the empirically best action in MABs applies also to MaxBRUE, resulting in the best known convergence rates among online MCTS algorithms to date. Moreover, in line with MAB results, the superiority of MaxBRUE with a *permissive budget* as well as the superiority of the UCB( $\alpha$ )-based exploration algorithms MaxUCT and MpaUCT with a *moderate budget* has been demonstrated empirically. We have also shown that a particular exploration control mechanism applied to MaxBRUE substantially improves its performance. We believe that this mechanism and variations of it can be valuable to other online planning algorithms as well. Finally, other exploration strategies that are found appealing in the context of MABs can also be “converted” to MDPs following the lines of this work.

**Acknowledgements** This work was partially supported by the EOARD grant FA8655-12-1-2096, and the ISF grant 1045/12.

## References

Balla, R., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*, 40–45.

- Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding Klondike Solitaire with Monte-Carlo planning. In *ICAPS*.
- Bonet, B., and Geffner, H. 2012. Action selection for MDPs: Anytime AO\* vs. UCT. In *AAAI*.
- Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte-Carlo tree search methods. *IEEE Trans. on Comp. Intell. and AI in Games* 143.
- Bubeck, S., and Munos, R. 2010. Open loop optimistic planning. In *COLT*, 477–489.
- Bubeck, S.; Munos, R.; and Stoltz, G. 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.* 412(19):1832–1852.
- Busoniu, L., and Munos, R. 2012. Optimistic planning for Markov decision processes. In *AISTATS*, number 22 in JMLR (Proceedings Track), 182–189.
- Cazenave, T. 2009. Nested Monte-Carlo search. In *IJCAI*, 456–461.
- Coquelin, P.-A., and Munos, R. 2007. Bandit algorithms for tree search. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 67–74.
- Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the Canadian Traveler’s problem. In *AAAI*.
- Feldman, Z., and Domshlak, C. 2012. Simple regret optimization in online planning for Markov decision processes. *CoRR arXiv:1206.3382v2 [cs.AI]*.
- Feldman, Z., and Domshlak, C. 2013. Monte-Carlo planning: Theoretically fast convergence meets practical efficiency. In *UAI*.
- Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *AIJ* 175(11):1856–1875.
- Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49(2-3):193–208.
- Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *ICAPS*, 135–143.
- Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *ECML*, 282–293.
- Kolobov, A.; Mausam; and Weld, D. 2012. LRTDP vs. UCT for online probabilistic planning. In *AAAI*.
- Péret, L., and Garcia, F. 2004. On-line search for solving Markov decision processes via heuristic sampling. In *ECAI*, 530–534.
- Puterman, M. 1994. *Markov Decision Processes*. Wiley.
- Robbins, H. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58(5):527535.
- Rosin, C. D. 2011. Nested rollout policy adaptation for Monte Carlo tree search. In *IJCAI*, 649–654.
- Sturtevant, N. 2008. An analysis of UCT in multi-player games. In *CCG*, 3749.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.
- Tolpin, D., and Shimony, S. E. 2012. MCTS based on simple regret. In *AAAI*.

# Landmarks in Oversubscription Planning

Vitaly Mirkis and Carmel Domshlak<sup>1</sup>

**Abstract.** In the basic setup of oversubscription planning (OSP), the objective is to achieve an as valuable as possible subset of goals within a fixed allowance of the total action cost [32]. Continuing from the recent successes in exploiting logical goal-reachability landmarks in classical planning, we develop a framework for exploiting such landmarks in heuristic-search OSP. We show how standard landmarks of certain classical planning tasks can be compiled into the OSP task of interest, resulting in an equivalent OSP task with a lower budget, and thus with a smaller search space. We then show how such landmark-based task enrichment can be combined in a mutually stratifying way with the *BFBB* search used for OSP planning. Our empirical evaluation confirms the effectiveness of the proposed landmark-based budget reduction scheme.

## 1 INTRODUCTION

In most general terms, deterministic planning is a problem of finding paths in large-scale yet concisely represented state-transition systems. In what these days is called classical planning [11], the task is to find an as *cost-effective* path as possible to a *goal-satisfying* state. In contrast, in what Smith [32] baptized as “oversubscription” planning (OSP), the task is to find an as *goal-effective* (or *valuable*) state as possible via a *cost-satisfying* path. In other words, the hard constraint of classical planning translates to only preference in OSP, and the hard constraint of OSP translates to only preference in classical planning. Finally, in “optimal” classical planning and OSP, the tasks are further constrained to finding only *most* cost-effective paths and *most* goal-effective states, respectively.

Classical planning and OSP constitute the most fundamental variants of deterministic planning, with many other variants of deterministic planning being defined in terms of mixing and relaxing the two. For instance, “net-benefit” planning tries to achieve both (classical) cost-effectiveness of the path and (OSP) goal-effectiveness of the end-state by additively combining the two measures, but at the same time, it relaxes the hard constraints of (classical) goal-satisfaction and (OSP) cost-satisfaction [31, 1, 4, 2, 7, 24]. Another popular setup is “cost-bounded” planning, in which both (classical) goal-satisfaction and (OSP) cost-satisfaction are pursued, but both (classical) cost-effectiveness of the path and (OSP) goal-effectiveness of the end-state are relaxed/ignored [33, 34, 13, 15, 19, 12, 27].

While OSP has been advocated over the years on par with classical planning, so far, the theory and practice of the latter have been studied and advanced much more intensively. The remarkable success and continuing progress of heuristic-search solvers for classical planning is one notable example. Primary enablers of this success are the advances in domain-independent approximations, or heuristics, of the cost needed to achieve a goal state from a given state.

With our focus here on optimal planning, two classes of approximation techniques have been found especially useful in the context of optimal classical planning: those based on state-space *abstractions* [10, 14, 18, 22] and these based on logical *landmarks* for goal reachability [21, 17, 9, 5, 28].

Considering OSP as heuristic search, a question is then whether some similar-in-spirit (yet possibly mathematically different) approximation techniques can be developed for heuristic-search OSP. Recently, the authors provided the first affirmative answer to this question in the context of abstractions by developing the actual notion of OSP abstractions, investigating the complexity of working with them for the purpose of heuristic approximation, and demonstrating empirically that using OSP abstraction heuristics within a best-first branch-and-bound (*BFBB*) search can be extremely effective in practice [25]. In contrast, the prospects of goal-reachability landmarks in heuristic-search OSP have not been investigated yet.

This is precisely the contribution of this paper: First, we introduce and study  $\varepsilon$ -landmarks, the logical properties of OSP plans that achieve valuable states. We show that  $\varepsilon$ -landmarks correspond to regular landmarks of certain classical planning tasks that can be (straightforwardly) derived from the OSP tasks of interest. We then show how such  $\varepsilon$ -landmarks can be compiled into the OSP task of interest, resulting in an equivalent OSP task, but with a stricter cost satisfaction constraint, and thus with a smaller effective search space. Finally, we show how such landmark-based task enrichment can be combined in a mutually stratifying way with the *BFBB* search used for OSP planning, resulting in an incremental procedure that interleaves search and landmark discovery. The entire framework is independent of the OSP planner specifics, and in particular, of the heuristic functions it employs. Our empirical evaluation on a large set of OSP tasks confirms the effectiveness of the proposed approach.

## 2 PRELIMINARIES

Since both OSP and classical planning tasks are discussed in the paper, we use a formalism that is based on the standard STRIPS formalism with non-negative operator costs (cf. [17]), extended to OSP in line with the notation of our earlier paper on OSP [25].

**Planning Tasks.** A *planning task structure* is given by a pair  $\langle V, O \rangle$ , where  $V$  is a finite set of propositional state variables, and  $O$  is a finite set of operators. State variables are also called propositions or facts. A state  $s \in 2^V$  is a subset of facts, representing the propositions which are currently true. Each operator  $o \in O$  is associated with preconditions  $\text{pre}(o) \subseteq V$ , add effects  $\text{add}(o) \subseteq V$ , and delete effects  $\text{del}(o) \subseteq V$ . Applying an operator  $o$  in  $s$  results in state  $(s \setminus \text{del}(o)) \cup \text{add}(o)$ , which we denote as  $s[o]$ . The notation is only defined if  $o$  is applicable in  $s$ , i.e., if  $\text{pre}(o) \subseteq s$ . Applying a sequence  $\langle o_1, \dots, o_k \rangle$  of operators to a state is defined inductively as  $s[\epsilon] := s$  and  $s[\langle o_1, \dots, o_k \rangle] := (s[\langle o_1, \dots, o_{k-1} \rangle])[o_k]$ .

<sup>1</sup> Technion, Haifa, Israel, emails: {mirkis@tx}{dcarmel@ie}.technion.ac.il

A *classical planning task*  $\Pi = \langle V, O; I, G, cost \rangle$  extends its structure  $\langle V, O \rangle$  with an initial state  $I \subseteq V$ , a goal  $G \subseteq V$ , and a real-valued, nonnegative operator cost function  $cost : O \rightarrow \mathbb{R}^{0+}$ . An operator sequence  $\pi$  is called an  $s$ -plan if it is applicable in  $s$ , and  $G \subseteq s[\pi]$ . The cost of  $s$ -plan  $\pi$  is  $cost(\pi) := \sum_{o \in \pi} cost(o)$ , and  $\pi$  is optimal if its cost is minimal among all  $s$ -plans. The objective in classical planning is to find an  $I$ -plan of as low cost as possible or prove that no  $I$ -plan exists. Optimal classical planning is devoted to searching for optimal  $I$ -plans only.

An *oversubscription planning (OSP) task*  $\Pi = \langle V, O; I, cost, u, b \rangle$  extends its structure  $\langle V, O \rangle$  with four components: an initial state  $I \subseteq V$  and an operator cost function  $cost : O \rightarrow \mathbb{R}^{0+}$  as above, plus a succinctly represented and efficiently computable state value function  $u : S \rightarrow \mathbb{R}^{0+}$ , and a cost budget  $b \in \mathbb{R}^{0+}$ . In what follows, we assume  $u(s) = \sum_{v \in s} u(v)$ , i.e., the value of state  $s$  is the sum of (mutually independent) values of propositions which are true in  $s$ . Conceptually, our results equally apply to general value functions, but the complexity of certain construction steps may vary between different families of value functions.

In OSP, an operator sequence  $\pi$  is called an  $s$ -plan if it is applicable in  $s$ , and  $\sum_{o \in \pi} cost(o) \leq b$ . While even an empty operator sequence is an  $s$ -plan for any state  $s$ , the objective in OSP is to find an  $I$ -plan that achieves as valuable a state as possible. By  $\hat{u}(\pi)$  we refer to the value of the end-state of  $\pi$ , that is,  $\hat{u}(\pi) = u(s[\pi])$ . Optimal OSP is devoted to searching for optimal  $I$ -plans only: An  $s$ -plan  $\pi$  is optimal if  $\hat{u}(\pi)$  is maximal among all the  $s$ -plans.

**Heuristics.** The two major ingredients of any heuristic-search planner are its search algorithm and heuristic function. In classical planning, the heuristic is typically a function  $h : 2^V \rightarrow \mathbb{R}^{0+} \cup \{\infty\}$ , with  $h(s)$  estimating the cost  $h^*(s)$  of optimal  $s$ -plans. A heuristic  $h$  is admissible if it is *lower-bounding*, i.e.,  $h(s) \leq h^*(s)$  for all states  $s$ . All common heuristic search algorithms for optimal classical planning, such as  $A^*$ , require admissible heuristics.

In OSP, a heuristic is a function  $h : 2^V \times \mathbb{R}^{0+} \rightarrow \mathbb{R}^{0+}$ , with  $h(s, b)$  estimating the value  $h^*(s, b)$  of optimal  $s$ -plans under cost budget  $b$ . A heuristic  $h$  is admissible if it is *upper-bounding*, i.e.,  $h(s, b) \geq h^*(s, b)$  for all states  $s$  and all cost budgets  $b$ . Here as well, search algorithms for optimal OSP, such as best-first branch-and-bound (BFBB) discussed later on in detail, require admissible heuristics.

**Landmarks in Classical Planning.** For a state  $s$  in a classical planning task  $\Pi$ , a landmark is a property of operator sequences that is satisfied by all  $s$ -plans [20]. For instance, a fact landmark for a state  $s$  is a fact that is true at some point in every  $s$ -plan. Several admissible landmark heuristics have been shown as extremely effective in optimal classical planning [21, 17, 5, 28]. These heuristics use extended notions of landmarks which are subsumed by *disjunctive action landmarks*. Each such landmark is a set of operators such that every  $s$ -plan contains at least one action from that set. In what follows we consider this popular notion of landmarks, and simply refer to disjunctive action landmarks for a state  $s$  as  $s$ -landmarks. For ease of presentation, most of our discussion will take place in the context of landmarks for the initial state of the task, and these will simply be referred to as *landmarks* (for  $\Pi$ ).

Deciding whether an operator set  $L \subseteq O$  is a landmark for classical planning task  $\Pi$  is PSPACE-hard [29]. Therefore, all landmark heuristics employ methods for landmark discovery that are polynomial-time, sound, but incomplete. In what follows we assume access to such a procedure; the actual way the landmarks are discovered is tangential to our contribution. For a set  $\mathcal{L}$  of  $s$ -

landmarks, a *landmark cost function*  $lcost : \mathcal{L} \rightarrow \mathbb{R}^{0+}$  is admissible if  $\sum_{L \in \mathcal{L}} lcost(L) \leq h^*(s)$ . For a singleton set  $\mathcal{L} = \{L\}$ ,  $lcost(L) := \min_{o \in L} cost(o)$  is a natural admissible landmark cost function, and it extends directly to non-singleton sets of pairwise disjoint landmarks. For more general sets of landmarks,  $lcost$  can be devised (in polynomial time) via operator cost partitioning [23], either given  $\mathcal{L}$  [21], or within the actual process of generating  $\mathcal{L}$  [17].

### 3 “BRING ME SOMETHING” LANDMARKS

While landmarks play an important role in (both satisficing and optimal) classical planning, so far they have not been exploited in OSP. At first glance, this is probably no surprise: Since landmarks must hold in all plans, and the empty operator sequence is always a plan for any OSP task, the notion of landmark does not seem useful here.

Having said that, consider the anytime “output improvement” property of the forward-search branch-and-bound algorithms used for heuristic-search OSP. The empty plan is not interesting there not only because it is useless, but also because it is “found” by the search algorithm right at the get-go. In general, at all stages of the search, anytime algorithms like BFBB maintain the best-so-far solution  $\pi$ , and prune all branches that promise value lower or equal to  $\hat{u}(\pi)$ . Hence, in principle, such algorithms may benefit from information about properties that are “satisfied by all plans with value larger than  $x$ .” Unfortunately, it is not yet clear how the machinery for discovering classical planning landmarks can be adapted to discovery of such “value landmarks” while preserving polynomial-time complexity on general OSPs and arbitrary lower bounds  $x$ .

Looking at what is needed and what is available, our goal here is to exploit this machinery as it is. While the value of different  $s$ -plans in an OSP task  $\Pi$  varies between zero and the value of the optimal  $s$ -plan (which may also be zero), let an  $\varepsilon$ -landmark for state  $s$  be any property that is satisfied by any  $s$ -plan  $\pi$  that achieves *something valuable*. For instance, with the disjunctive action landmarks we use here, if  $L \subseteq O$  is an  $\varepsilon$ -landmark for  $s$ , then every  $s$ -plan  $\pi$  with  $\hat{u}(\pi) > 0$  contains an operator from  $L$ . In what follows, unless stated otherwise, we focus on  $\varepsilon$ -landmarks for (the initial state of)  $\Pi$ .

Given an OSP task  $\Pi = \langle V, O; I, cost, u, b \rangle$ , let a classical planning task  $\Pi_\varepsilon = \langle V_\varepsilon, O_\varepsilon; I_\varepsilon, cost_\varepsilon, G_\varepsilon \rangle$  be constructed as  $V_\varepsilon = V \cup \{g\}$ ,  $I_\varepsilon = I$ ,  $G_\varepsilon = \{g\}$ , and  $O_\varepsilon = O \cup O_g$ , where, for each proposition  $v$  with  $u(v) > 0$ ,  $O_g$  contains an operator  $o_v$  with  $pre(o_v) = \{v\}$ ,  $add(o_v) = \{g\}$ ,  $del(o_v) = \emptyset$ , and  $cost_\varepsilon(o_v) = 0$ . For all the original operators  $o \in O$ ,  $cost_\varepsilon(o) = cost(o)$ . In other words,  $\Pi_\varepsilon$  extends the structure of  $\Pi$  with a set of zero-cost actions such that applying any of them indicates achieving a positive value in  $\Pi$ . In what follows, we refer to  $\Pi_\varepsilon$  as the  $\varepsilon$ -*compilation* of  $\Pi$ . Constructing  $\Pi_\varepsilon$  from  $\Pi$  is trivially polynomial time, and

**Theorem 1** *For any OSP task  $\Pi$ , any landmark  $L$  for  $\Pi_\varepsilon$  such that  $L \subseteq O$  is an  $\varepsilon$ -landmark for  $\Pi$ .*

With Theorem 1 in hand,<sup>2</sup> we can now derive  $\varepsilon$ -landmarks for  $\Pi$  using any method for classical planning landmark extraction, such as that employed by the LAMA planner [30] or the LM-Cut family of techniques [17, 5]. However, at first glance, the discriminative power of knowing “what is needed to achieve *something valuable*” seems to be negligible when it comes to deriving effective heuristic estimates for OSP. The good news is that, in OSP, such information can be effectively exploited in a slightly different way.

<sup>2</sup> Due to space limitations, all proofs are delegated to a full technical report [26].

```

BFBB ( $\Pi = \langle V, O; I, cost, u, b \rangle$ )
  open := new max-heap ordered by  $f(n) = h(s[n], b - g(n))$ 
  open.insert(make-root-node( $I$ ))
  closed :=  $\emptyset$ ; best-cost :=  $\emptyset$ 
  initialize best solution  $n^* := I$ 
  while not open.empty()
     $n := \text{open.pop-max}()$ 
    if  $f(n) \leq u(s[n^*])$ : break
    if  $s[n] \notin \text{closed}$  or  $g(n) < \text{best-cost}(s[n])$ :
      closed := closed  $\cup \{s[n]\}$ 
      best-cost( $s[n]$ ) :=  $g(n)$ 
      foreach  $o \in O(s[n])$ :
         $n' := \text{make-node}(s[n][o])$ 
        if  $g(n') > b$  or  $f(n') \leq u(s[n^*])$ : continue
        if  $u(s[n']) > u(s[n^*])$ : update  $n^* := n'$ 
        open.insert( $n'$ )
  return  $n^*$ 

```

Figure 1. Best-first branch-and-bound (BFBB) search for OSP

### 3.1 $\varepsilon$ -Landmarks and Budget Reduction

In the same way that  $A^*$  constitutes a canonical heuristic-search algorithm for optimal classical planning, anytime *best-first branch-and-bound* (BFBB) probably constitutes such an algorithm for optimal OSP.<sup>3</sup> Figure 1 depicts a pseudo-code description of BFBB.  $s[n]$  there denotes the state associated with search node  $n$ . In BFBB for OSP, a node  $n$  with maximum evaluation function  $h(s[n], b - g(n))$  is selected from the OPEN list. The duplicate detection and reopening mechanisms in BFBB are similar to those in  $A^*$ . In addition, BFBB maintains the best solution  $n^*$  found so far and uses it to prune all generated nodes evaluated no higher than  $u(s[n^*])$ . Likewise, complying with the semantics of OSP, all generated nodes  $n$  with cost-so-far  $g(n)$  higher than the problem’s budget  $b$  are also immediately pruned. When the OPEN list becomes empty or the node  $n$  selected from the list promises less than the lower bound, BFBB returns (the plan associated with) the best solution  $n^*$ , and if  $h$  is admissible, i.e., the  $h$ -based pruning of the generated nodes is sound, then the returned plan is guaranteed to be optimal.

Now, consider a schematic example of searching for an optimal plan for an OPS task  $\Pi$  with budget  $b$ , using BFBB with an admissible heuristic  $h$ . Suppose that there is only one sequence of (all unit-cost) operators,  $\pi = \langle o_1, o_2, \dots, o_{b+1} \rangle$ , applicable in the initial state of  $\Pi$ , and that the only positive value state along  $\pi$  is its end-state. While clearly no value higher than zero can be achieved in  $\Pi$  under the given budget of  $b$ , the search will continue beyond the initial state, unless  $h(I, \cdot)$  counts the cost of all the  $b + 1$  actions of  $\pi$ . Now, suppose that  $h(I, \cdot)$  counts only the cost of  $\{o_i, \dots, o_{b+1}\}$  for some  $i > 0$ , but  $\{o_1\}, \{o_2\}, \dots, \{o_{i-1}\}$  are all discovered to be  $\varepsilon$ -landmarks for  $\Pi$ . Given that, suppose that we modify  $\Pi$  by (a) setting the cost of operators  $o_1, o_2, \dots, o_{i-1}$  to zero, and (b) *reducing the budget to  $b - i + 1$* . This modification seems to preserve the semantics of  $\Pi$ , while on the modified task, BFBB with the same heuristic  $h$  will prune the initial state and thus establish without any search that the empty plan is an optimal plan for  $\Pi$ . Of course, the way  $\Pi$  is modified in this example is as simplistic as the example itself. Yet, this example does motivate the idea of *landmark-based budget reduction* for OSP, as well as illustrates the basic idea behind the *generically sound* task modifications that we discuss next.

Let  $\Pi = \langle V, O; I, cost, u, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ , and  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ . Given that, a new OSP task  $\Pi_{\mathcal{L}} = \langle V_{\mathcal{L}}, O_{\mathcal{L}}; I_{\mathcal{L}}, cost_{\mathcal{L}}, u_{\mathcal{L}}, b_{\mathcal{L}} \rangle$  with budget  $b_{\mathcal{L}} =$

```

compile-and-BFBB ( $\Pi = \langle V, O; I, cost, u, b \rangle$ )
   $\Pi_{\varepsilon} := \varepsilon$ -compilation of  $\Pi$ 
   $\mathcal{L} :=$  a set of landmarks for  $\Pi_{\varepsilon}$ 
   $lcost :=$  admissible landmark cost function for  $\mathcal{L}$ 
   $\Pi_{\mathcal{L}} :=$  budget reducing compilation of  $(\mathcal{L}, lcost)$  into  $\Pi$ 
   $n^* := \text{BFBB}(\Pi_{\mathcal{L}})$ 
  return plan for  $\Pi$  associated with  $n^*$ 

```

Figure 2. BFBB search with landmark-based budget reduction

$b - \sum_{i=1}^n lcost(L_i)$  is constructed as follows. The set of variables  $V_{\mathcal{L}} = V \cup \{v_{L_1}, \dots, v_{L_n}\}$  extends  $V$  with a new proposition per  $\varepsilon$ -landmark in  $\mathcal{L}$ . These new propositions are all initially true, and  $I_{\mathcal{L}} = I \cup \{v_{L_1}, \dots, v_{L_n}\}$ . The value function  $u_{\mathcal{L}} = u$  remains unchanged—the new propositions do not affect the value of the states. Finally, the operator set is extended as  $O_{\mathcal{L}} = O \cup \bigcup_{i=1}^n O_{L_i}$ , with  $O_{L_i}$  containing an operator  $\bar{o}$  for each  $o \in L_i$ , with  $\text{pre}(\bar{o}) = \text{pre}(o) \cup \{v_{L_i}\}$ ,  $\text{add}(\bar{o}) = \text{add}(o)$ ,  $\text{del}(\bar{o}) = \text{del}(o) \cup \{v_{L_i}\}$ , and, importantly,  $cost_{\mathcal{L}}(\bar{o}) = cost(o) - lcost(L_i)$ . In other words,  $\Pi_{\mathcal{L}}$  extends the structure of  $\Pi$  by mirroring the operators of each  $\varepsilon$ -landmark  $L_i$  with their “ $lcost(L_i)$  cheaper” versions, while ensuring that these cheaper operators can be applied no more than once along an operator sequence from the initial state. At the same time, introduction of these discounted operators for  $L_i$  is compensated for by reducing the budget by precisely  $lcost(L_i)$ , leading to effective equivalence between  $\Pi$  and  $\Pi_{\mathcal{L}}$ .

**Theorem 2** *Let  $\Pi = \langle V, O; I, cost, u, b \rangle$  be an OSP task,  $\mathcal{L}$  be a set of pairwise disjoint  $\varepsilon$ -landmarks for  $\Pi$ ,  $lcost$  be an admissible landmark cost function from  $\mathcal{L}$ , and  $\Pi_{\mathcal{L}}$  be the respective budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $\hat{u}(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}}$  for  $\Pi_{\mathcal{L}}$  with  $\hat{u}(\pi_{\mathcal{L}}) = \hat{u}(\pi)$ , and vice versa.*

The above budget reducing compilation of  $\Pi$  to  $\Pi_{\mathcal{L}}$  is clearly polynomial time. Putting things together, we can see that the compile-and-BFBB procedure depicted in Figure 2 (1) generates an  $\varepsilon$ -compilation  $\Pi_{\varepsilon}$  of  $\Pi$ , (2) uses off-the-shelf tools for classical planning to generate a set of landmarks  $\mathcal{L}$  for  $\Pi_{\varepsilon}$  and an admissible landmark cost function  $lcost$ , and (3) compiles  $(\mathcal{L}, lcost)$  into  $\Pi$ , obtaining an OSP task  $\Pi_{\mathcal{L}}$ . The optimal solution for  $\Pi_{\mathcal{L}}$  (and thus for  $\Pi$ ) is then searched for using a search algorithm for optimal OSP such as BFBB.

Before we proceed to consider more general sets of landmarks, a few comments concerning the setup of Theorem 2 are now probably in place. First, if the reduced budget  $b_{\mathcal{L}}$  turns out to be lower than the cost of the cheapest action applicable in the initial state, then no search is needed, and the empty plan can be reported as optimal right away. Second, zero-cost landmarks are useless in our compilation as much as they are useless in deriving landmark heuristics for optimal planning. Hence,  $lcost$  in what follows is assumed to be strictly positive. Third, having both  $o$  and  $\bar{o}$  applicable at a state of  $\Pi_{\varepsilon}$  brings no benefits yet adds branching to the search. Hence, in our implementation, for each landmark  $L_i \in \mathcal{L}$  and each operator  $o \in L_i$ , the precondition of  $o$  in  $O_{\mathcal{L}}$  is extended with  $\{\neg v_{L_i}\}$ . It is not hard to verify that this extension<sup>4</sup> preserves the correctness of  $\Pi_{\mathcal{L}}$  in terms of Theorem 2. Finally, if the value of the initial state is not zero, that is, the empty plan has some positive value, then  $\varepsilon$ -compilation  $\Pi_{\varepsilon}$  of  $\Pi$  will have no positive cost landmarks at all. However, this can easily be fixed by considering as “valuable” only facts  $v$  such that both  $u(v) > 0$  and  $v \notin I$ . For now we put this difficulty aside and assume that  $\hat{u}(\epsilon) = 0$ . Later, however, we come back to consider it more systematically.

<sup>3</sup> BFBB is also extensively used for net-benefit planning [3, 7, 8], as well as some other variants of deterministic planning [4, 6].

<sup>4</sup> This modification requires augmenting our STRIPS-like formalism with negative preconditions, but this augmentation is straightforward.

### 3.2 Non-Disjoint $\varepsilon$ -Landmarks

While the compilation  $\Pi_{\mathcal{L}}$  above is sound for pairwise disjoint landmarks, this is not so for more general sets of  $\varepsilon$ -landmarks. For example, consider a planning task  $\Pi$  in which, for some operator  $o$ , we have  $\text{cost}(o) = b$ ,  $\hat{u}(\langle o \rangle) > 0$ , and  $\hat{u}(\pi) = 0$  for all other operator sequences  $\pi \neq \langle o \rangle$ . That is, a value greater than zero is achievable in  $\Pi$ , but only via the operator  $o$ . Suppose now that our set of  $\varepsilon$ -landmarks for  $\Pi$  is  $\mathcal{L} = \{L_1, \dots, L_n\}$ ,  $n > 1$ , and that all of these  $\varepsilon$ -landmarks contain  $o$ . In this case, while the budget in  $\Pi_{\mathcal{L}}$  is  $b_{\mathcal{L}} = b - \sum_{i=1}^n \text{lcost}(L_i)$ , the cost of the cheapest replica  $\bar{o}$  of  $o$ , that is, the cost of the cheapest operator sequence achieving a non-zero value in  $\Pi$ , is  $\text{cost}(o) - \min_{i=1}^n \text{lcost}(L_i) > b_{\mathcal{L}}$ . Hence, no state with positive value will be reachable from  $I_{\mathcal{L}}$  in  $\Pi_{\mathcal{L}}$ , and thus  $\Pi$  and  $\Pi_{\mathcal{L}}$  are not “value equivalent” in the sense of Theorem 2.

Since non-disjoint landmarks can bring more information, and they are typical to outputs of standard techniques for landmark extraction in classical planning, we now present a different, slightly more involved, compilation that is both polynomial and sound for arbitrary sets of  $\varepsilon$ -landmarks. Let  $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$  be an OSP task,  $\mathcal{L} = \{L_1, \dots, L_n\}$  be a set of  $\varepsilon$ -landmarks for  $\Pi$ , and  $\text{lcost}$  be an admissible landmark cost function from  $\mathcal{L}$ . For each operator  $o$ , let  $\mathcal{L}(o)$  denote the set of all landmarks in  $\mathcal{L}$  that contain  $o$ . Given that, a new OSP task  $\Pi_{\mathcal{L}^*} = \langle V_{\mathcal{L}^*}, O_{\mathcal{L}^*}; I_{\mathcal{L}^*}, \text{cost}_{\mathcal{L}^*}, u_{\mathcal{L}^*}, b_{\mathcal{L}^*} \rangle$  is constructed as follows. Similarly to  $\Pi_{\mathcal{L}}$ , we have  $b_{\mathcal{L}^*} = b - \sum_{i=1}^n \text{lcost}(L_i)$ ,  $V_{\mathcal{L}^*} = V \cup \{v_{L_1}, \dots, v_{L_n}\}$ ,  $I_{\mathcal{L}^*} = I \cup \{v_{L_1}, \dots, v_{L_n}\}$ , and  $u_{\mathcal{L}^*} = u$ . The operator set  $O_{\mathcal{L}^*}$  extends  $O$  with two sets of operators:

- For each operator  $o \in O$  that participates in some landmark from  $\mathcal{L}$ ,  $O_{\mathcal{L}^*}$  contains an action  $\bar{o}$  with  $\text{pre}(\bar{o}) = \text{pre}(o) \cup \{v_L \mid L \in \mathcal{L}(o)\}$ ,  $\text{add}(\bar{o}) = \text{add}(o)$ ,  $\text{del}(\bar{o}) = \text{del}(o) \cup \{v_L \mid L \in \mathcal{L}(o)\}$ ,  $\text{cost}_{\mathcal{L}^*}(\bar{o}) = \text{cost}(o) - \sum_{L \in \mathcal{L}(o)} \text{lcost}(L)$ .
- For each  $L \in \mathcal{L}$ ,  $O_{\mathcal{L}^*}$  contains an action  $\text{get}(L)$  with  $\text{pre}(\text{get}(L)) = \{-v_L\}$ ,  $\text{add}(\text{get}(L)) = \{v_L\}$ ,  $\text{del}(\text{get}(L)) = \emptyset$ ,  $\text{cost}_{\mathcal{L}^*}(\text{get}(L)) = \text{lcost}(L)$ .

For example, let  $\mathcal{L} = \{L_1, L_2, L_3\}$ ,  $L_1 = \{a, b\}$ ,  $L_2 = \{b, c\}$ ,  $L_3 = \{a, c\}$ , with all operators having the cost of 2, and let  $\text{lcost}(L_1) = \text{lcost}(L_2) = \text{lcost}(L_3) = 1$ . In  $\Pi_{\mathcal{L}^*}$ , we have  $V_{\mathcal{L}^*} = V \cup \{v_{L_1}, v_{L_2}, v_{L_3}\}$  and  $O_{\mathcal{L}^*} = O \cup \{\bar{a}, \bar{b}, \bar{c}, \text{get}(L_1), \text{get}(L_2), \text{get}(L_3)\}$ , with, e.g.,  $\text{pre}(\bar{a}) = \text{pre}(a) \cup \{v_{L_1}, v_{L_3}\}$ ,  $\text{add}(\bar{a}) = \text{add}(a)$ ,  $\text{del}(\bar{a}) = \text{del}(a) \cup \{v_{L_1}, v_{L_3}\}$ , and  $\text{cost}_{\mathcal{L}^*}(\bar{a}) = 0$ , and, for  $\text{get}(L_1)$ ,  $\text{pre}(\text{get}(L_1)) = \text{del}(\text{get}(L_1)) = \emptyset$ ,  $\text{add}(\text{get}(L_1)) = \{v_{L_1}\}$ , and  $\text{cost}_{\mathcal{L}^*}(\text{get}(L_1)) = 1$ .

**Theorem 3** Let  $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$  be an OSP task and  $\Pi_{\mathcal{L}^*}$  a budget reducing compilation of  $\Pi$ . For every  $\pi$  for  $\Pi$  with  $\hat{u}(\pi) > 0$ , there is a plan  $\pi_{\mathcal{L}^*}$  for  $\Pi_{\mathcal{L}^*}$  with  $\hat{u}(\pi_{\mathcal{L}^*}) = \hat{u}(\pi)$ , and vice versa.

### 4 $\varepsilon$ -LANDMARKS & INCREMENTAL BFBB

As we discussed earlier, if the value of the initial state is not zero, i.e., the empty plan has some positive value, then the basic  $\varepsilon$ -compilation  $\Pi_{\varepsilon}$  of  $\Pi$  will have no positive cost landmarks at all. In passing we noted that this small problem can be remedied by considering as “valuable” only facts  $v$  such that both  $u(v) > 0$  and  $v \notin I$ . We now consider this aspect of OSP more closely, and show how  $\varepsilon$ -landmarks discovery and incremental revelation of plans by BFBB can be combined in a mutually stratifying way.

Let  $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$  be the OSP task of our interest, and suppose we are given a set of plans  $\pi_1, \dots, \pi_n$  for  $\Pi$ . If so, then we are no longer interested in searching for plans that “achieve something,” but in searching for plans that achieve *something beyond*

inc-compile-and-BFBB ( $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$ )

```

initialize global variables:
   $n^* := I$  // best solution so far
   $S_{\text{ref}} := \{I\}$  // current reference states
loop:
   $\Pi_{(\varepsilon, S_{\text{ref}})} = (\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ 
   $\mathcal{L} :=$  a set of landmarks for  $\Pi_{(\varepsilon, S_{\text{ref}})}$ 
   $\text{lcost} :=$  admissible landmark cost function from  $\mathcal{L}$ 
   $\Pi_{\mathcal{L}^*} :=$  budget reducing compilation of  $(\mathcal{L}, \text{lcost})$  into  $\Pi$ 
  if inc-BFBB( $\Pi_{\mathcal{L}^*}, S_{\text{ref}}, n^*$ ) = done:
    return plan for  $\Pi$  associated with  $n^*$ 

```

```

inc-BFBB( $\Pi, S_{\text{ref}}, n^*$ )
open := new max-heap ordered by  $f(n) = h(s[n], b - g(n))$ 
open.insert(make-root-node( $I$ ))
closed :=  $\emptyset$  best-cost :=  $\emptyset$ ;
while not open.empty():
   $n :=$  open.pop-max()
  if goods( $s[n]$ )  $\not\subseteq$  goods( $s'$ ) for all  $s' \in S_{\text{ref}}$ :
     $S_{\text{ref}} := S_{\text{ref}} \cup \{s[n]\}$ 
    if termination criterion: return updated
  if  $f(n) \leq u(s[n])$ : break
  // ...
  // similar to BFBB in Figure 1
return done

```

Figure 3. Iterative BFBB with landmark enhancement

what  $\pi_1, \dots, \pi_n$  already achieve. For  $1 \leq i \leq n$ , let  $s_i = I[\pi_i]$  be the end-state of  $\pi_i$ , and for any set of propositions  $s \subseteq V$ , let  $\text{goods}(s) \subseteq s$  be the set of all facts  $v \in s$  such that  $u(v) > 0$ . If a plan  $\pi$  with end-state  $s$  achieves something beyond what  $\pi_1, \dots, \pi_n$  already achieve, then  $\text{goods}(s) \setminus \text{goods}(s_i) \neq \emptyset$  for all  $1 \leq i \leq n$ .

We now put this observation to work. Given an OSP task  $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$  and a set of reference states  $S_{\text{ref}} = \{s_1, \dots, s_n\}$  of  $\Pi$ , let a classical planning task  $\Pi_{(\varepsilon, S_{\text{ref}})} = \langle V_{\varepsilon}, O_{\varepsilon}; I_{\varepsilon}, G_{\varepsilon}, \text{cost}_{\varepsilon} \rangle$  be constructed as follows. The variable set  $V_{\varepsilon} = V \cup \{x_1, \dots, x_n, \text{search}, \text{collect}\}$  extends  $V$  with a new proposition per state in  $S_{\text{ref}}$ , plus two auxiliary control variables. In the initial state, all the new variables but *search* are false, i.e.,  $I_{\varepsilon} = I \cup \{\text{search}\}$ , and the goal is  $G_{\varepsilon} = \{x_1, \dots, x_n\}$ . The operator set  $O_{\varepsilon}$  contains three sets of operators: First, each operator  $o \in O$  is represented in  $O_{\varepsilon}$  by an operator  $\bar{o}$ , with the only difference between  $o$  and  $\bar{o}$  (including cost) being that  $\text{pre}(\bar{o}) = \text{pre}(o) \cup \{\text{search}\}$ . We denote this set of new operators  $\bar{O}$  by  $\bar{O}$ . Second, for each  $s_i \in S_{\text{ref}}$  and each value-carrying fact  $g$  that is *not* in  $s_i$ , i.e., for each  $g \in \text{goods}(V) \setminus s_i$ ,  $O_{\varepsilon}$  contains a zero-cost action  $o_{i,g}$  with  $\text{pre}(o_{i,g}) = \{g, \text{collect}\}$ ,  $\text{add}(o_{i,g}) = \{x_i\}$ ,  $\text{del}(o_{i,g}) = \emptyset$ . Finally,  $O_{\varepsilon}$  contains a zero-cost action *finish* with  $\text{pre}(\text{finish}) = \emptyset$ ,  $\text{del}(\text{finish}) = \{\text{search}\}$ , and  $\text{add}(\text{finish}) = \{\text{collect}\}$ .

It is easy to verify that (1) the goal  $G_{\varepsilon}$  cannot be achieved without applying the *finish* operator, (2) the operators  $\bar{o}$  can be applied only before *finish*, and (3) the subgoal achieving operators  $o_{i,g}$  can be applied only after *finish*. Hence, the first part of any plan for  $\Pi_{(\varepsilon, S_{\text{ref}})}$  determines a plan for  $\Pi$ , and the second part “verifies” that the end-state of that plan achieves a subset of value-carrying propositions  $\text{goods}(V)$  that is included in no state from  $S_{\text{ref}}$ .<sup>5</sup>

**Theorem 4** Let  $\Pi = \langle V, O; I, \text{cost}, u, b \rangle$  be an OSP task,  $S_{\text{ref}} = \{s_1, \dots, s_n\} \subseteq 2^V$  be a subset of  $\Pi$ ’s states, and  $L$  be a landmark for  $\Pi_{(\varepsilon, S_{\text{ref}})}$  such that  $L \subseteq \bar{O}$ . For any plan  $\pi$  for  $\Pi$  such that  $\text{goods}(I[\pi]) \setminus \text{goods}(s_i) \neq \emptyset$  for all  $s_i \in S_{\text{ref}}$ ,  $\pi$  contains an instance of at least one operator from  $L' = \{o \mid \bar{o} \in L\}$ .

Theorem 4 allows us to define an iterative version of BFBB, successive iterations of which correspond to running the regular BFBB

<sup>5</sup> This “plan in two parts” technique appears to be helpful in many planning formalism compilations; see, e.g., [24].



on successively more informed  $(\varepsilon, S_{\text{ref}})$ -compilations of  $\Pi$ , with the states discovered at iteration  $i$  making the  $(\varepsilon, S_{\text{ref}})$ -compilation used at iteration  $i + 1$  more informed. The respective procedure *inc-compile-and-BFBB* is depicted in Figure 3. This procedure maintains a set of reference states  $S_{\text{ref}}$  and the best solution so far  $n^*$ , and loops over calls to *inc-BFBB*, a modified version of *BFBB*. At each iteration of the loop, *inc-BFBB* is called with an  $(\varepsilon, S_{\text{ref}})$ -compilation of  $\Pi$ , created on the basis of the *current*  $S_{\text{ref}}$  and  $n^*$ , and it is provided with access to both  $S_{\text{ref}}$  and  $n^*$ . The reference set  $S_{\text{ref}}$  is then extended by *inc-BFBB* with all the non-redundant value-carrying states discovered during the search, and  $n^*$  is updated if the search discovers nodes of higher value.

If and when the OPEN list becomes empty or the node  $n$  selected from the list promises less than the lower bound, *inc-BFBB* returns an indicator, *done*, that the best solution  $n^*$  found so far, across the iterations of *inc-compile-and-BFBB*, is optimal. In that case, *inc-compile-and-BFBB* leaves its loop and extracts that optimal plan from  $n^*$ . However, *inc-BFBB* may also terminate in a different way, if a certain complementary termination criterion is satisfied. The latter criterion comes to assess whether the updates to  $S_{\text{ref}}$  performed in the current session of *BFBB* warrant updating the  $(\varepsilon, S_{\text{ref}})$ -compilation and restarting the search.<sup>6</sup> If terminated this way, *inc-BFBB* returns a respective indicator, and *inc-compile-and-BFBB* goes into another iteration of its loop, with the *updated*  $S_{\text{ref}}$  and  $n^*$ .

## 5 EMPIRICAL EVALUATION

We have implemented a prototype heuristic-search OSP solver on top of the Fast Downward planner [16]. The implementation included<sup>7</sup>:

- $(\varepsilon, S_{\text{ref}})$ -compilation of OSP tasks  $\Pi$ ;
- Generation of disjunctive action landmarks for  $(\varepsilon, S_{\text{ref}})$ -compilations using the LM-Cut procedure [17] of Fast Downward;
- The incremental *BFBB* procedure *inc-compile-and-BFBB* from the previous section, with the search termination criterion being satisfied (only) if the examined node  $n$  improves over current value lower bound; and
- An additive abstraction heuristic from the framework of Mirkis and Domshlak [25], incorporating (i) an ad hoc action cost partition over  $k$  projections of the planning task onto connected subsets of ancestors of the respective  $k$  goal variables in the causal graph, and (ii) a value partition that associates the value of each goal (only) with the respective projection. The size of each projection was limited to 1000 abstract states.

After some preliminary evaluation, we also added two (optimality preserving) enhancements to the search. First, the auxiliary variables of our compilations increase the dimensionality of the problem, and this is well known to negatively affect the quality of the projection abstractions. Hence, we devised the projections with respect to the *original* OSP problem  $\Pi$ , and the open list was ordered as if the search is done on the original problem, that is, by  $h(s[n]^{\downarrow V}, b - g(n) + \sum_{v_L \notin s[n]} lcost(L))$ , where  $s[n]^{\downarrow V}$  is the projection of the state  $s[n]$  on the variables of the original OSP task  $\Pi$ . This change in heuristic evaluation is sound, as Theorem 3 in particular implies that any admissible heuristic for  $\Pi$  is also an admissible heuristic for  $\Pi_{\mathcal{L}^*}$ , and vice versa. Second, when a new node  $n$  is generated, we check whether  $g(n) + \sum_{v_L \notin s[n]} lcost(L) \geq$

<sup>6</sup> While the optimality of the algorithm holds for *any* such termination condition, the latter should greatly affect the runtime efficiency of the algorithm.

<sup>7</sup> We are not aware of any other domain-independent planner for optimal OSP.

$g(n') + \sum_{v_L \notin s[n']} lcost(L)$ , for some previously generated node  $n'$  that corresponds to the same state of the original problem  $\Pi$ , i.e.,  $s[n']^{\downarrow V} = s[n]^{\downarrow V}$ . If so, then  $n$  is pruned right away. Optimality preservation of this enhancement is established in [26].

Since, unlike classical and net-benefit planning, OSP lacks standard benchmarks for comparative evaluation, we have cast in this role the STRIPS classical planning domains from the International Planning Competitions (IPC) 1998-2006. This “translation” to OSP was done by associating a separate unit-value with each sub-goal. The evaluation included the regular *BFBB* planning for  $\Pi$ , solving  $\Pi$  using landmark-based compilation via *compile-and-BFBB*, and the straightforward setting of *inc-compile-and-BFBB* described above. All three approaches were evaluated under the blind heuristic and the additive abstraction heuristic as above.

Figure 4 depicts the results of our evaluation in terms of expanded nodes on all the aforementioned IPC tasks for which we could determine offline the minimal cost needed to achieve all the goals in the task. Each task was approached under four different budgets, corresponding to 25%, 50%, 75%, and 100% of the minimal cost needed to achieve all the goals in the task, and each run was restricted to 10 minutes. Figures 4(a) and 4(b) compare the performance of *BFBB* and *compile-and-BFBB* with blind (a) and abstraction (b) heuristics. Figures 4(c) and 4(d) provide a similar comparison between *BFBB* and *inc-compile-and-BFBB*.<sup>8</sup>

As Figure 4 shows, the results are satisfactory. With no informative heuristic guidance at all, the number of nodes expanded by *compile-and-BFBB* was typically much lower than the number of nodes expanded by *BFBB*, with the difference reaching three orders of magnitude more than once. Of the 760 task/budget pairs behind Figure 4a, 81 pairs were solved by *compile-and-BFBB* with no search at all (by proving that no plan can achieve value higher than that of the initial state), while, unsurprisingly, only 4 of these tasks were solved with no search by *BFBB*.

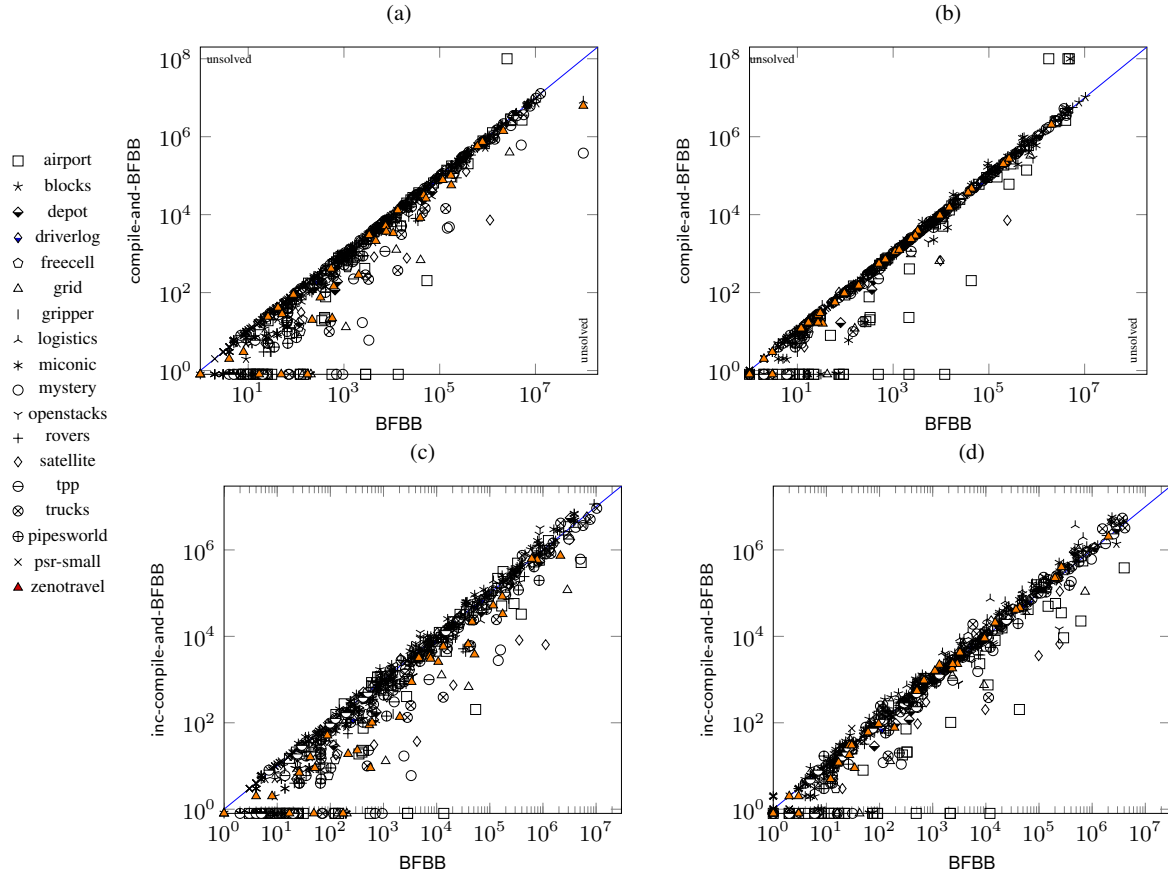
As expected, the value of landmark-based budget reduction is lower when the search is equipped with a meaningful heuristic (Figure 4b). Yet, even with our abstraction heuristic in hand, the number of nodes expanded by *compile-and-BFBB* was often substantially lower than the number of nodes expanded by *BFBB*. Here, *BFBB* and *compile-and-BFBB* solved with no search 39 and 85 task/budget pairs, respectively. Finally, despite the rather ad hoc setting of our incremental *inc-compile-and-BFBB* procedure, switching from *compile-and-BFBB* to *inc-compile-and-BFBB* was typically beneficial, though much deeper investigation and development of *inc-compile-and-BFBB* is obviously still required.

**ACKNOWLEDGMENTS** This work was partially supported by the ISF grant 1045/12, and the EOARD grant FA8655-12-1-2096.

## REFERENCES

- [1] J. A. Baier, F. Bacchus, and S. A. McIlraith, ‘A heuristic search approach to planning with temporally extended preferences’, in *IJCAI*, pp. 1808–1815, (2007).
- [2] J. Benton, M. Do, and S. Kambhampati, ‘Anytime heuristic search for partial satisfaction planning’, *AIJ*, 173(5-6), 562–592, (2009).
- [3] J. Benton, M. van den Briel, and S. Kambhampati, ‘A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems’, in *ICAPS*, pp. 34–41, (2007).
- [4] B. Bonet and H. Geffner, ‘Heuristics for planning with penalties and rewards formulated in logic and computed through circuits’, *AIJ*, 172(12-13), 1579–1604, (2008).

<sup>8</sup> We do not compare here between the running times, but the per-node CPU time overhead due to landmark-based budget reduction was  $\leq 10\%$ .



**Figure 4.** Comparative view of empirical results in terms of expanded nodes: (a) & (b) *BFBB* vs. *compile-and-BFBB*, (c) & (d) *BFBB* vs. *inc-compile-and-BFBB*, with blind ((a) & (c)) and additive projections ((b) & (d)) heuristics

- [5] B. Bonet and M. Helmert, ‘Strengthening landmark heuristics via hitting sets’, in *ECAI*, pp. 329–334, (2010).
- [6] R. Brafman and Y. Chernyavsky, ‘Planning with goal preferences and constraints’, in *ICAPS*, pp. 182–191, Monterey, CA, (2005).
- [7] A. J. Coles and A. Coles, ‘LPRPG-P: Relaxed plan heuristics for planning with preferences’, in *ICAPS*, (2011).
- [8] M. B. Do, J. Benton, M. van den Briel, and S. Kambhampati, ‘Planning with goal utility dependencies’, in *IJCAI*, pp. 1872–1878, (2007).
- [9] C. Domshlak, M. Katz, and S. Edelkamp, ‘Landmark-enhanced abstraction heuristics’, *AIJ*, **189**, 48–68, (2012).
- [10] S. Edelkamp, ‘Planning with pattern databases’, in *ECP*, pp. 13–24, (2001).
- [11] R. E. Fikes and N. Nilsson, ‘STRIPS: A new approach to the application of theorem proving to problem solving’, *AIJ*, **2**, 189–208, (1971).
- [12] A. Gerevini, A. Saetti, and I. Serina, ‘An approach to efficient planning with numerical fluents and multi-criteria plan quality’, *AIJ*, **172**(8-9), 899–944, (2008).
- [13] P. Haslum, ‘Heuristics for bounded-cost search’, in *ICAPS*, (2013).
- [14] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig, ‘Domain-independent construction of pattern database heuristics for cost-optimal planning’, in *AAAI*, pp. 1007–1012, (2007).
- [15] P. Haslum and H. Geffner, ‘Heuristic planning with time and resources’, in *ECP*, (2001).
- [16] M. Helmert, ‘The Fast Downward planning system’, *JAIR*, **26**, 191–246, (2006).
- [17] M. Helmert and C. Domshlak, ‘Landmarks, critical paths and abstractions: What’s the difference anyway?’, in *ICAPS*, pp. 162–169, (2009).
- [18] M. Helmert, P. Haslum, and J. Hoffmann, ‘Flexible abstraction heuristics for optimal sequential planning’, in *ICAPS*, pp. 200–207, (2007).
- [19] J. Hoffmann, C. P. Gomes, B. Selman, and H. A. Kautz, ‘SAT encodings of state-space reachability problems in numeric domains’, in *IJCAI*, pp. 1918–1923, (2007).
- [20] J. Hoffmann, J. Porteous, and L. Sebastia, ‘Ordered landmarks in planning’, *JAIR*, **22**, 215–278, (2004).
- [21] E. Karpas and C. Domshlak, ‘Cost-optimal planning with landmarks’, in *IJCAI*, pp. 1728–1733, (2009).
- [22] M. Katz and C. Domshlak, ‘Implicit abstraction heuristics’, *JAIR*, **39**, 51–126, (2010).
- [23] M. Katz and C. Domshlak, ‘Optimal admissible composition of abstraction heuristics’, *AIJ*, **174**, 767–798, (2010).
- [24] E. Keyder and H. Geffner, ‘Soft goals can be compiled away’, *JAIR*, **36**, 547–556, (2009).
- [25] V. Mirkis and C. Domshlak, ‘Abstractions for oversubscription planning’, in *ICAPS*, (2013).
- [26] V. Mirkis and C. Domshlak, ‘Landmarks in oversubscription planning’, Technical Report IE/IS-2014-01, Technion, (2014).
- [27] H. Nakhost, J. Hoffmann, and M. Müller, ‘Resource-constrained planning: A Monte Carlo random walk approach’, in *ICAPS*, (2012).
- [28] F. Pommerening and M. Helmert, ‘Incremental LM-Cut’, in *ICAPS*, (2013).
- [29] J. Porteous, L. Sebastia, and J. Hoffmann, ‘On the extraction, ordering, and usage of landmarks in planning’, in *ECP*, (2001).
- [30] S. Richter, M. Helmert, and M. Westphal, ‘Landmarks revisited’, in *AAAI*, pp. 975–982, (2008).
- [31] R. Sanchez and S. Kambhampati, ‘Planning graph heuristics for selecting objectives in over-subscription planning problems’, in *ICAPS*, pp. 192–201, (2005).
- [32] D. Smith, ‘Choosing objectives in over-subscription planning’, in *ICAPS*, pp. 393–401, (2004).
- [33] J. T. Thayer and W. Ruml, ‘Bounded suboptimal search: A direct approach using inadmissible estimates’, in *IJCAI*, pp. 674–679, (2011).
- [34] J. T. Thayer, R. T. Stern, A. Felner, and W. Ruml, ‘Faster bounded-cost search using inadmissible estimates’, in *ICAPS*, (2012).

# Abstractions for Oversubscription Planning

Vitaly Mirkis and Carmel Domshlak

Technion - Israel Institute of Technology

Haifa, Israel

{dcarmel,mirkis}@tx.technion.ac.il

## Abstract

In deterministic OSP, the objective is to achieve an as valuable as possible subset of goals within a fixed allowance of the total action cost. Although numerous applications in various fields share this objective, no substantial algorithmic advances have been made beyond the very special settings of net-benefit optimization. Tracing the key sources of progress in classical planning, we identify a severe lack of domain-independent approximations for OSP, and start with investigating the prospects of abstraction approximations for this problem. In particular, we define the notion of additive abstractions for OSP, study the complexity of deriving effective abstractions from a rich space of hypotheses, and reveal some substantial, empirically relevant islands of tractability.

## Introduction

In deterministic planning, the basic structure of acting with underconstrained or overconstrained resources is respectively captured by *classical* planning and *oversubscription* planning. In classical planning, all goals must be achieved at as low a total cost of the actions as possible. In oversubscription planning (OSP), an as valuable as possible subset of goals should be achieved within a fixed allowance of the total action cost. While both theory and practice of classical planning have been rapidly advancing, progress in OSP has been mostly in the direction of net-benefit planning. In net-benefit planning, no explicit restriction is put on the plan cost, and the action costs and goal utilities are assumed to be comparable, with the objective being maximizing the difference between the cumulative value of the achieved goals and the cost invested in achieving them. Although there are numerous interesting algorithms for net-benefit planning, it was recently shown to be polynomial-time reducible to classical planning (Keyder and Geffner 2009). As such, it constitutes an extremely special variant of oversubscription.

A closer look shows that the recent progress in classical planning stems, to a large extent, from advances in domain-independent approximations, or heuristics, of the cost needed to achieve all the goals from a given state. It is thus possible that having a similarly rich palette of effective heuristic functions for OSP would advance the state-of-the-art in that problem. In principle, the reduction of Keyder

and Geffner (2009) from net-benefit to classical planning can be used to reduce OSP to classical planning with numeric state variables (Fox and Long 2003; Helmert 2002). So far, however, progress in classical planning with numeric state variables has mostly been achieved along delete relaxation heuristics (Hoffmann 2003; Edelkamp 2003), and these heuristics do not preserve information on consumable resources: the “negative” action effects that decrease the values of numeric variables are ignored, possibly up to some special handling of so-called “cyclic resource transfer” (Coles et al. 2008).

In this work we make first steps towards effective heuristics for OSP, and in particular, towards admissible *abstraction heuristics* for this problem. In classical planning, state-space abstractions are among the most prominent techniques for devising admissible heuristics (Edelkamp 2002; Haslum et al. 2007; Helmert, Haslum, and Hoffmann 2007; Katz and Domshlak 2010a). Departing from the most basic question of what state-space abstractions for OSP actually are (and what they are not), we show that the very notion of abstraction substantially differs in classical and in OSP. We define additive abstractions and abstraction heuristics for OSP, and investigate computational complexity of deriving effective abstraction heuristics in the scope of homomorphic abstraction skeletons, paired with cost, value, and budget partitions. Along with revealing some significant islands of tractability, we expose an interesting interplay between knapsack-style problems, convex optimization, and principles borrowed from explicit abstractions for classical planning. We believe that this interplay opens the road to much further research.

## Formalism and Background

In line with the SAS<sup>+</sup> formalism for deterministic planning (Bäckström and Klein 1991; Bäckström and Nebel 1995), a *planning task structure* is given by a pair  $\langle V, A \rangle$ , where  $V$  is a set of  $n$  finite-domain *state variables*, and  $A$  is a finite set of *actions*. Each complete assignment to  $V$  is called a *state*, and  $S = \text{dom}(v_1) \times \dots \times \text{dom}(v_n)$  is the *state space* of the structure  $\langle V, A \rangle$ . Each action  $a$  is a pair  $\langle \text{pre}(a), \text{eff}(a) \rangle$  of partial assignments to  $V$  called *preconditions* and *effects*, respectively. Denoting by  $\mathcal{V}(p) \subseteq V$  the subset of variables instantiated by a partial assignment  $p$ , action  $a$  is applicable in a state  $s$  iff  $s[\mathcal{V}(p)] = \text{pre}(a)[\mathcal{V}(p)]$  for

all  $v \in \mathcal{V}(\text{pre}(a))$ . Applying  $a$  changes the value of each  $v \in \mathcal{V}(\text{eff}(a))$  to  $\text{eff}(a)[v]$ . The resulting state is denoted by  $s[a]$ ; by  $s[\langle a_1, \dots, a_k \rangle]$  we denote the state obtained from sequential application of the (applicable in turn) actions  $a_1, \dots, a_k$  starting at state  $s$ .

In classical planning, a planning task  $\Pi = \langle V, A; s_0, G, c \rangle$  extends its structure with an initial state  $s_0 \in S$ , a goal specification  $G$ , typically modeled as a partial assignment to  $V$ , and an action cost function  $c : A \rightarrow \mathbb{R}^{0+}$ . An action sequence  $\rho$  is called an  $s$ -plan if it is applicable in  $s$ , and  $G \subseteq s[\rho]$ . An  $s$ -plan is optimal if the sum of its action costs is minimal among all  $s$ -plans. The objective in classical planning is to find an  $s_0$ -plan of as low cost as possible, with optimal classical planning being devoted to searching for optimal  $s_0$ -plans only.

In contrast, a **oversubscription planning (OSP) task**  $\Pi = \langle V, A; s_0, c, u, b \rangle$  extends its structure with four components: an *initial state*  $s_0 \in S$  and an *action cost function*  $c : A \rightarrow \mathbb{R}^{0+}$  as above, plus a succinctly represented and efficiently computable *state value function*  $u : S \rightarrow \mathbb{R}^{0+}$ , and a *cost budget*  $b \in \mathbb{R}^{0+}$ . An action sequence  $\rho$  is called an  $s$ -plan if it is applicable in  $s$ , and  $\sum_{a \in \rho} c(a) \leq b$ ; by  $\hat{u}(\rho)$  we refer to the value of the end-state of  $\rho$ , that is,  $\hat{u}(\rho) = u(s[\rho])$ . While empty action sequence is an  $s$ -plan for any state  $s$ , the objective in oversubscription planning is to find an  $s_0$ -plan that achieves as valuable a state as possible, and **optimal oversubscription planning** is devoted to searching for optimal  $s_0$ -plans only: An  $s$ -plan  $\rho$  is *optimal* if  $\hat{u}(\rho)$  is maximal among all the  $s$ -plans, and if  $\rho$  is optimal, then  $h^*(s) \stackrel{\text{def}}{=} \hat{u}(\rho)$ .

Each planning task  $\Pi$  induces a state-transition model, or transition graph. Following Katz and Domshlak (2010b), we distinguish between the actual node/edge-weighted transition graphs, and their weights-omitted, qualitative skeletons, referred to as transition graph structures. Informally, the latter capture the dynamics of the planning tasks, while the former associate these dynamics with “performance measures” (Russell and Norvig 2009). A **transition graph structure** (or **tg-structure**) is a triplet  $\mathcal{T} = \langle S, L, Tr \rangle$  where  $S$  is the finite set of states,  $L$  is the finite set of labels, and  $Tr \subseteq S \times L \times S$  is a set of labeled state transitions. Each tg-structure  $\mathcal{T} = \langle S, L, Tr \rangle$  implicitly defines a **space of performance measures** that can be associated with it. In the context of OSP, this space constitutes  $C \times U \times B$  where  $C$  is the set of all functions from labels  $L$  to  $\mathbb{R}^{0+}$ ,  $U$  is the set of all functions from states  $S$  to  $\mathbb{R}^{0+}$ , and  $B = \mathbb{R}^{0+}$ . A **transition graph** (or **t-graph**)  $\Phi = \langle \mathcal{T}, c, u, b \rangle$  associates a tg-structure  $\mathcal{T}$  with a specific performance measure  $(c, u, b) \in C \times U \times B$ . A path from state  $s$  along the transitions of  $\mathcal{T}$  is an  $s$ -plan for  $\Phi$  if  $\sum_{(s, l, s') \in \pi} c(l) \leq b$ .

The tg-structure  $\mathcal{T}(\Pi)$  induced by a planning task  $\Pi = \langle V, A; s_0, c, u, b \rangle$  is induced by the structure  $\langle V, A \rangle$  of the latter: the states and labels of  $\mathcal{T}(\Pi)$  are states  $S = \text{dom}(V)$  and actions  $A$  of  $\Pi$ , respectively, and  $(s, a, s[a]) \in Tr$  iff action  $a$  is applicable in state  $s$ . The t-graph induced by a planning task  $\Pi = \langle V, A; s_0, c, u, b \rangle$  is  $\Phi(\Pi) = \langle \mathcal{T}(\Pi), c, u, b \rangle$ . Since there is an obvious correspondence between the  $s$ -plans for  $\Pi$  and the  $s$ -plans for  $\Phi(\Pi)$ , search-

ing in  $\Phi(\Pi)$  corresponds to planning for  $\Pi$  via state-space search, and heuristic-search such procedures employ heuristic functions to estimate the relative attractiveness of various parts of the t-graph  $\Phi(\Pi)$ . A useful heuristic function must be both efficiently computable from the planning task, as well as relatively accurate in its estimates. Improving the accuracy of a heuristic function without substantially worsening the time complexity of computing it translates into faster search for plans.

In classical planning, numerous approximation techniques, such as monotonic relaxation, critical trees, logical landmarks, and abstractions, have been translated to extremely useful heuristic functions, and different heuristics for classical planning can also be combined into their point-wise maximizing and/or additive ensembles.<sup>1</sup> Unfortunately, while some of these ideas have also been translated to classical planning with numeric state variables, the resulting heuristics do not appear useful for OSP. Approaching the need for effective heuristics for OSP, here we focus on abstractions for OSP, from their very definition and properties, to the prospects of deriving (admissible) abstraction heuristics.

## Abstractions for OSP

The term “abstraction” is usually associated with simplifying the original system, factoring out details less crucial in the given context. In classical planning, the abstract t-graphs are required not to increase the distances between the (abstracted) states (Katz and Domshlak 2010b), and such “distance conservation” is in particular guaranteed by homomorphic abstractions, obtained by systematically contracting sets of states into single abstract states (Helmert, Haslum, and Hoffmann 2007). In turn, an additive abstraction in classical planning is a set of abstractions, inter-constrained to *jointly* not overestimate the state-to-state costs of the original task. As we now show, the concept of (additive) abstractions in OSP is very different, and, for better and for worse, has many more degrees of freedom than the respective concept in classical planning.

For  $k \in \mathbb{N}^+$ , by  $[k]$  we denote the set  $\{1, 2, \dots, k\}$ . Let  $\mathcal{T} = \langle S, L, Tr \rangle$  be a tg-structure, and let  $\mathcal{T}_i = \langle S_i, L_i, Tr_i \rangle$ ,  $i \in [k]$ , be a set of some tg-structures, each related to  $\mathcal{T}$  via some state mapping  $\alpha_i : S \rightarrow S_i$ . Such a set of tg-structure/state-mapping pairs  $\mathcal{AS} = \{(\mathcal{T}_i, \alpha_i)\}_{i \in [k]}$  is what is called an **abstraction skeleton** for  $\mathcal{T}$  (Katz and Domshlak 2010b). Now, if  $C_i \times U_i \times B_i$  is the performance measure space of  $\mathcal{T}_i$ , then  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ , with  $\mathbf{C} = \times C_i$ ,  $\mathbf{U} = \times U_i$ , and  $\mathbf{B} = \times B_i$ , is the **joint performance measure space of  $\mathcal{AS}$** . That is, any choice of  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  induces a set of t-graphs  $\{\langle \mathcal{T}_i, \mathbf{c}[i], \mathbf{u}[i], \mathbf{b}[i] \rangle\}_{i \in [k]}$ . In turn, once  $\mathcal{T}$  is associated with a performance measure  $(c, u, b)$ , each joint performance measure  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$  of  $\mathcal{AS}$  either does or does not constitute an (additive) abstraction of the t-graph  $\Phi = \langle \mathcal{T}, c, u, b \rangle$ . In Definition 1 we capture this relation at even a more refined level—with respect to a specific state of

<sup>1</sup>For a comparative survey and pointers to the literature, we refer the reader to Helmert and Domshlak (2009).

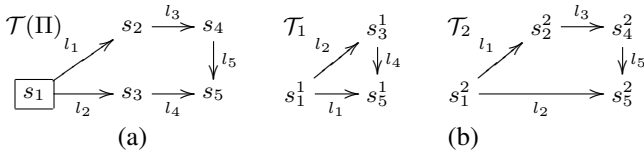


Figure 1: Illustration for our running example

interest in  $\Phi$ —and we do that directly in terms of t-graphs induced by OSP tasks.

**Definition 1 (Additive Abstraction)**

Let  $\Pi = \langle V, A; s, c, u, b \rangle$  be an OSP task,  $\mathcal{AS} = \{(\mathcal{T}_i, \alpha_i)\}_{i \in [k]}$  be an abstraction skeleton for  $\mathcal{T}(\Pi)$ , and  $(c, u, b)$  be a joint performance measure for  $\mathcal{AS}$ . The set of t-graphs  $\mathcal{A}_{(c, u, b)} = \{(\mathcal{T}_i, c[i], u[i], b[i])\}_{i \in [k]}$  is an **(additive) abstraction for  $\Pi$** , denoted by  $\mathcal{A}_{(c, u, b)} \in_s \mathcal{AS}$ , if

$$h^*(s) \leq h_{\mathcal{A}_{(c, u, b)}}(s) \stackrel{\text{def}}{=} \sum_{i \in [k]} h_i^*(\alpha_i(s)),$$

i.e., when  $h_{\mathcal{A}_{(c, u, b)}}(s)$  is an admissible estimate of  $h^*(s)$ .

In simple terms, a set of abstractions in OSP is constrained to jointly *not underestimate* the value that can be obtained from a *concrete state* of the original task within a given cost budget. For example, let  $\mathcal{T} = \langle \{s_i\}_{i \in [5]}, \{l_i\}_{i \in [5]}, Tr \rangle$  in Figure 1a be a tg-structure of some OSP task  $\Pi$  with initial state  $s_1$ , and  $\mathcal{AS} = \{(\mathcal{T}_1, \alpha_1), (\mathcal{T}_2, \alpha_2)\}$ , with tg-structures  $\mathcal{T}_1, \mathcal{T}_2$  as in Figure 1b and state mappings

$$\alpha_1(s_i) = \begin{cases} s_5^1, & i \in \{2, 4\} \\ s_1^1, & \text{otherwise} \end{cases} \quad \alpha_2(s_i) = \begin{cases} s_5^2, & i = 3 \\ s_2^2, & \text{otherwise} \end{cases}.$$

Let t-graphs  $\Phi(\Pi) = \langle \mathcal{T}(\Pi), c, u, b \rangle$ ,  $\Phi_1 = \langle \mathcal{T}_1, c_1, u_1, b_1 \rangle$ ,  $\Phi_2 = \langle \mathcal{T}_2, c_2, u_2, b_2 \rangle$  be defined via label cost functions  $c, c_1, c_2$  that associate all labels with a cost of 1, budgets  $b = b_1 = b_2 = 2$ , and state value functions  $u, u_1, u_2$  that evaluate to zero on all states except for  $s_5, s_5^1, s_5^2$ , on which they respectively evaluate to one. Considering the state  $s_1$  of  $\Pi$ , the optimal  $s_1$ -plan for  $\Pi$  is  $\pi = \langle (s_1, l_2, s_3), (s_3, l_4, s_5) \rangle$  with  $\hat{u}(\pi) = 1$ . The optimal  $\alpha_1(s_1)$ -plan for  $\Phi_1$  is  $\pi_1 = \langle (s_1^1, l_1, s_5^1) \rangle$  with  $\hat{u}_1(\pi_1) = 1$ , and the optimal  $\alpha_2(s_1)$ -plan for  $\Phi_2$  is  $\pi_2 = \langle (s_1^2, l_2, s_5^2) \rangle$ , with  $\hat{u}_2(\pi_2) = 1$ . Since  $\hat{u}(\pi) \leq \hat{u}_1(\pi_1) + \hat{u}_2(\pi_2)$ ,  $\mathcal{A} = \{\Phi_1, \Phi_2\}$  is an additive abstraction for  $\Pi$ .

**Theorem 1** For any OSP task  $\Pi = \langle V, A; s, c, u, b \rangle$ , any abstraction skeleton  $\mathcal{AS}$  of  $\mathcal{T}(\Pi)$ , and any  $\mathcal{A} \in_s \mathcal{AS}$ , if the t-graphs of  $\mathcal{A}$  are given explicitly, then  $h_{\mathcal{A}}(s)$  can be computed in time polynomial in  $\|\Pi\|$  and  $\|\mathcal{A}\|$ .

The proof is straightforward: Let  $\mathcal{A} = \{\Phi_i\}_{i \in [k]}$ , with  $\Phi_i = \langle \mathcal{T}_i, c_i, u_i, b_i \rangle$ , be an additive abstraction for  $\Pi$ . For  $i \in [k]$ , let  $S'_i = \{s' \in S_i \mid c_i(\alpha_i(s), s') \leq b_i\}$ . Since  $\mathcal{A}$  is given explicitly, computing shortest paths from  $\alpha_i(s)$  to all states in  $\mathcal{T}_i$ , and thus computing  $S'_i$ , can be done in time polynomial in  $\|\mathcal{A}\|$  for all  $i \in [k]$ . If  $\pi_i$  is an optimal  $\alpha_i(s)$ -plan for  $\Phi_i$ , then by Definition 1,  $\hat{u}_i(\pi_i) = \max_{s' \in S'_i} u_i(s')$ ,

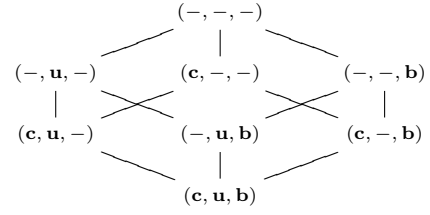


Figure 2: Fragments of restricted optimization over  $\mathbf{A}(s)$ .

and thus computing  $h_{\mathcal{A}}(s) = \sum_{i \in [k]} \hat{u}_i(\pi_i)$  is polynomial time in  $\|\mathcal{A}\|$ .

While Theorem 1 is positive, it establishes only a necessary condition for the relevance of OSP abstractions to practice. Given an OSP task  $\Pi$ , and having fixed an abstraction skeleton  $\mathcal{T}$  with a joint performance measure space  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ , for each state of interest  $s$ , we should be able to automatically identify an abstraction that provides us with as accurate (aka as *low*) an estimate as possible. Let  $\mathbf{A}(s) \subseteq \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  be the subset of joint performance measures that constitute abstractions for  $\Pi$ . Note that  $\mathbf{A}(s)$  is not a combinatorial rectangle in  $\mathbf{C} \times \mathbf{U} \times \mathbf{B}$ . For instance, consider t-graph  $\Phi(\Pi)$ , state  $s_1$  of  $\Phi(\Pi)$ , and abstraction skeleton  $\mathcal{AS}$  from our running example. Let  $c \in \mathbf{C}$  be a cost function vector with both  $c[1]$  and  $c[2]$  being constant, unit-cost functions, and two performance measures  $(c, u, b), (c, u', b') \in \mathbf{C} \times \mathbf{U} \times \mathbf{B}$  being defined via budget vectors  $b = \{b[1] = 2, b[2] = 0\}$  and  $b' = \{b'[1] = 0, b'[2] = 2\}$ , and value function vectors  $u$  and  $u'$ , with  $u[1], u[2], u'[1]$ , and  $u'[2]$  evaluating to zero on all states except for  $u[1](s_5^1) = u'[2](s_5^2) = 1$ . It is easy to verify that  $(c, u, b), (c, u', b') \in \mathbf{A}(s_1)$ , yet  $(c, u', b), (c, u, b') \notin \mathbf{A}(s_1)$ .

We now proceed with considering a specific family of additive abstractions, reveal some of its interesting properties, and show that it contains substantial islands of tractability. We break down and approach the overall agenda of complexity analysis of abstraction-based heuristic functions under *fixation* of some of the three dimensions of  $\mathbf{A}(s)$ : If, for instance, we are *given* a vector of value functions  $u$  that is *known* to belong to the projection of  $\mathbf{A}(s)$  on  $\mathbf{U}$ , then we can search for a quality abstraction from the abstraction subset  $H_{(-, u, -)}(s) \subset \mathbf{A}(s)$ , corresponding to the projection of  $\mathbf{A}(s)$  on  $\{u\}$ . As we show below, even some constrained estimate optimizations of this kind can be challenging. The lattice in Figure 2 depicts the range of options for such constrained optimization; at the extreme settings,  $H_{(-, -, -)}(s)$  is simply a renaming of  $\mathbf{A}(s)$ , and  $h_{(c, u, b)}(s)$  corresponds to a single abstraction  $(c, u, b) \in \mathbf{A}(s)$ .

## Partitions and Homomorphic Abstractions

With Definition 1 allowing for very general abstraction skeletons, in this work we focus on **homomorphic abstraction skeletons**<sup>2</sup>: Given a tg-structure  $\mathcal{T} = \langle S, L, Tr \rangle$ , an

<sup>2</sup>All the results also hold verbatim for the more general “labeled paths preserving” abstraction skeletons studied by Katz and Domshlak (2010b) in the context of optimal classical planning.

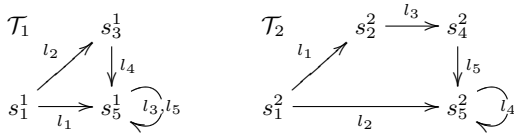


Figure 3: Homomorphic abstraction skeleton for  $\mathcal{T}(\Pi)$  in Figure 1.

abstraction skeleton  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$  of  $\mathcal{T}$  is homomorphic if, for  $i \in [k]$ ,  $L_i = L$ , and  $(s, l, s') \in Tr$  only if  $(\alpha_i(s), l, \alpha_i(s')) \in Tr_i$ . In our running example, the abstraction skeleton depicted in Figure 1b is not homomorphic, but its slight extension as in Figure 3, *ceteris paribus*, is homomorphic. Furthermore, we focus on a fragment of additive abstractions

$$\mathbf{A}_p(s) = \mathbf{A}(s) \cap [\mathbf{C}_p \times \mathbf{U}_p \times \mathbf{B}_p],$$

where  $\mathbf{C}_p \subseteq \mathbf{C}$ ,  $\mathbf{U}_p \subseteq \mathbf{U}$ , and  $\mathbf{B}_p \subseteq \mathbf{B}$  correspond to **cost**, **value**, and **budget partitions**, respectively. In what follows, by  $H_x^p$  we refer to  $H_x \cap \mathbf{A}_p(s)$ ; e.g.,  $H_{(-, \mathbf{u}, -)}^p = H_{(-, \mathbf{u}, -)}(s) \cap \mathbf{A}_p(s)$ . Given a t-graph  $\Phi = \langle \mathcal{T}, c, u, b \rangle$ , a homomorphic abstraction skeleton  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$  of  $\mathcal{T}$ , and  $\mathbf{c} \in \mathbf{C}$ , we have  $\mathbf{c} \in \mathbf{C}_p$  iff, for each label  $l$  in  $\mathcal{T}$ ,  $\sum_{i \in [k]} \mathbf{c}[i](l) \leq c(l)$ . Similarly,  $\mathbf{b} \in \mathbf{B}_p$  iff  $\sum_{i \in [k]} \mathbf{b}[i] \leq b$ , and (note the change in the direction of the inequality)  $\mathbf{u} \in \mathbf{U}_p$  iff, for each state  $s$  in  $\mathcal{T}$ ,  $\sum_{i \in [k]} \mathbf{u}[i](\alpha_i(s)) \geq u(s)$ .

Theorem 2 below establishes a “completeness” relationship between the sets  $\mathbf{C}_p$  and  $\mathbf{B}_p$ , as well as an even stronger “completeness” of  $\mathbf{C}_p$  and  $\mathbf{B}_p$ . In particular, it implies that, for all states  $s$ , the projections of  $\mathbf{A}_p(s)$  on  $\mathbf{C}_p$ ,  $\mathbf{U}_p$ , and  $\mathbf{B}_p$  are the entire sets  $\mathbf{C}_p$ ,  $\mathbf{U}_p$ , and  $\mathbf{B}_p$ , respectively.

**Theorem 2** *Given an OSP task  $\Pi = \langle V, A; s, c, u, b \rangle$  and a homomorphic abstraction skeleton  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$  of  $\mathcal{T}(\Pi)$ ,*

- (1) *for each action cost partition  $\mathbf{c} \in \mathbf{C}_p$ , there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  such that  $\mathcal{A}_{(\mathbf{c}, \mathbf{u}, \mathbf{b})} \in_s \mathcal{AS}$  for all  $\mathbf{u} \in \mathbf{U}_p$ .*
- (2) *for each budget partition  $\mathbf{b} \in \mathbf{B}_p$ , there exists an action cost partition  $\mathbf{c} \in \mathbf{C}_p$  such that  $\mathcal{A}_{(\mathbf{c}, \mathbf{u}, \mathbf{b})} \in_s \mathcal{AS}$  for all  $\mathbf{u} \in \mathbf{U}_p$ .*

*Proof:* Let  $\pi = \langle (s, a_1, s_1), (s_1, a_2, s_2), \dots, (s_{n-1}, a_n, s_n) \rangle$  be an optimal  $s$ -plan in  $\Phi(\Pi)$ . Given that  $\mathcal{AS}$  is homomorphic, let  $\pi_1, \dots, \pi_k$  be the projections of  $\pi$  on  $\mathcal{T}_1, \dots, \mathcal{T}_k$ , respectively, that is, for  $i \in [k]$ ,  $\pi_i = \langle (\alpha_i(s), a_1, \alpha_i(s_1)), \dots, (\alpha_i(s_{n-1}), a_n, \alpha_i(s_n)) \rangle$ .

(1) Let budget profile  $\mathbf{b}^* \in \mathbf{B}$  be defined as  $\mathbf{b}^*[i] = \sum_{j \in [n]} \mathbf{c}[i](a_j)$ , for  $i \in [k]$ . First, note that  $\mathbf{b}^* \in \mathbf{B}_p$  since

$$\sum_{i \in [k]} \mathbf{b}^*[i] = \sum_{i \in [k]} \sum_{j \in [n]} \mathbf{c}[i](a_j) \stackrel{(*)}{\leq} \sum_{j \in [n]} c(a_j) \stackrel{(**)}{\leq} b,$$

where  $(*)$  is by  $\mathbf{c}$  being an action cost partition, and  $(**)$  is by  $\pi$  being an  $s$ -plan for  $\Phi(\Pi)$ . Second, for any  $\mathbf{u} \in \mathbf{U}$ , by the construction of  $\mathbf{b}^*$ ,  $\pi_i$  is an  $\alpha_i(s)$ -plan for the t-graph  $\langle \mathcal{T}_i, \mathbf{c}[i], \mathbf{u}[i], \mathbf{b}^*[i] \rangle$ . Now, let  $\mathbf{u} \in \mathbf{U}_p$ , and for

$i \in [k]$ , let  $\pi_i^*$  be an optimal  $\alpha_i(s)$ -plan for that t-graph  $\langle \mathcal{T}_i, \mathbf{c}[i], \mathbf{u}[i], \mathbf{b}^*[i] \rangle$ . We have

$$\widehat{\mathbf{u}[i]}(\pi_i^*) \stackrel{(*)}{\geq} \widehat{\mathbf{u}[i]}(\pi_i) \stackrel{(**)}{\geq} \hat{u}(\pi), \quad (1)$$

where  $(*)$  is by optimality of  $\pi_i^*$ , and  $(**)$  is by  $\pi_i$  being the projection of  $\pi$  and  $\mathbf{u} \in \mathbf{U}_p$ . Therefore,  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  induces an additive abstraction for  $\Pi$ , that is,  $\mathcal{A}_{(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)} \in_s \mathcal{AS}$ .

(2) Let cost function profile  $\mathbf{c}^* \in \mathbf{C}$  be defined as  $\mathbf{c}^*[i](a) = c(a) \cdot \frac{\mathbf{b}[i]}{b}$ , for all actions  $a \in A$ , and all  $i \in [k]$ . First, we have  $\mathbf{c}^* \in \mathbf{C}_p$  since  $\mathbf{b} \in \mathbf{B}_p$  implies  $1/b \sum_{i \in [k]} \mathbf{b}[i] \in [0, 1]$ . Second, for any  $\mathbf{u} \in \mathbf{U}$ , by our construction of  $\mathbf{c}^*$ ,  $\pi_i$  is an  $\alpha_i(s)$ -plan for the t-graph  $\langle \mathcal{T}_i, \mathbf{c}^*[i], \mathbf{u}[i], \mathbf{b}[i] \rangle$ . Following now exactly the same line of reasoning as the one around Eq. 1 above accomplishes the proof that  $\mathcal{A}_{(\mathbf{c}^*, \mathbf{u}, \mathbf{b})} \in_s \mathcal{AS}$  for any  $\mathbf{u} \in \mathbf{U}_p$ .  $\square$

Again, an important corollary of Theorem 2 is that, for all states  $s$ , the projections of  $\mathbf{A}_p(s)$  on  $\mathbf{C}_p$ ,  $\mathbf{U}_p$ , and  $\mathbf{B}_p$  are the entire sets  $\mathbf{C}_p$ ,  $\mathbf{U}_p$ , and  $\mathbf{B}_p$ , respectively. A priori, this property should simplify the task of abstraction optimization, and later we show that this is indeed the case. However, complexity analysis of abstraction optimization in most general terms is still problematic because OSP formalism is parametric in the representation of value functions. Hence, as a first step, we restrict our attention to a fragment of  $\mathbf{A}_p(s)$  in which all abstract value functions are what we call 0-binary: A real-valued function  $f$  is a **0-binary** if it has image  $\text{img}(f) = \{0, v_f\}$  for some  $v_f \in \mathbb{R}$ . A set of 0-binary functions  $F$  is called **strong** if  $v_f = v_{f'}$  for all  $f, f' \in F$ . On the one hand, 0-binary functions constitute rather a basic family of value functions. Hence, if abstraction optimization is hard for them, it is likely to be hard for any non-trivial family of abstract value functions. On the other hand, 0-binary abstract value functions seem to fit well abstractions of planning tasks in which value functions are linear combinations of indicators, each representing achievement of a “goal value” for some state variable.

### $\mathbf{A}_p(s)$ and 0-Binary Value Partitions

Important roles in what follows are played by a well-known Knapsack problem, as well as some tools from convex optimization. In a Knapsack problem  $\langle \{w_i, \sigma_i\}_{i \in [n]}, W \rangle$ ,  $W$  is a weight allowance,  $[n]$  is a set of objects, and each  $i \in [n]$  has a weight  $w_i$  and a value  $\sigma_i$ . The objective is to find a subset  $X \subseteq [n]$  that maximizes  $\sum_{i \in X} \sigma_i$  over all subsets  $X' \subseteq [n]$  with  $\sum_{i \in X'} w_i \leq W$ . By *strict Knapsack* we refer to a variant of Knapsack in which that inequality constraint is strict. Knapsack is NP-hard, but there exist pseudo-polynomial algorithms for it that run in time polynomial in the description of the problem and in the unary representation of  $W$  (Garey and Johnson 1978). The latter property makes solving Knapsack practical in many applications where the ratio  $\frac{W}{\min_i w_i}$  is reasonably low. Likewise, if  $\sigma_i = \sigma_j$  for all  $i, j \in [n]$ , then a greedy algorithm solves the problem in linear time by iteratively expanding  $X$  by one of the weight-wise lightest objects in  $[n] \setminus X$ , until  $X$  cannot be expanded any further within  $W$ .

Let  $s$  be a state of an OSP task  $\Pi$ ,  $\mathcal{AS}$  be an (explicitly given) homomorphic abstraction skeleton of  $\mathcal{T}(\Pi)$ , and suppose that we fix a value partition  $\mathbf{u} \in \mathbf{U}_p$ . By Theorem 2,  $H_{(-,\mathbf{u},-)}^p(s)$  is not empty, and thus we can try computing  $\min_{(\mathbf{c},\mathbf{u},\mathbf{b}) \in H_{(-,\mathbf{u},-)}^p(s)} h_{(\mathbf{c},\mathbf{u},\mathbf{b})}(s)$ . As of yet, however, we do not know whether this task is polynomial-time solvable for any non-trivial class of value partitions. In fact, despite that  $H_{(-,\mathbf{u},-)}^p(s)$  is known to be non-empty, and so, too, are all of its subsets  $H_{(-,\mathbf{u},\mathbf{b})}^p(s)$  and  $H_{(\mathbf{c},\mathbf{u},-)}^p(s)$ , finding an abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in H_{(-,\mathbf{u},-)}^p(s)$  is not necessarily easy.

In that respect, our first tractability results are for abstraction discovery within  $H_{(-,\mathbf{u},-)}^p(s)$  where  $\mathbf{u}$  is a *strong* 0-binary value partition. The first (and the simpler) result in Theorem 3 further assumes a fixed action cost partition, while the next result, in Theorem 4, is on simultaneous selection of admissible pairs of cost and budget partitions. We also show how these results can be extended to pseudo-polynomial algorithms for *general* 0-binary value partitions.

### Theorem 3 ( $H_{(\mathbf{c},\mathbf{u},-)}^p(s)$ & strong 0-binary $\mathbf{u}$ )

Let  $\Pi = \langle V, A; s, c, u, b \rangle$  be an OSP task,  $\mathcal{AS}$  be an explicit homomorphic abstraction skeleton of  $\mathcal{T}(\Pi)$ , and  $\mathbf{u} \in \mathbf{U}_p$  be a strong 0-binary value partition. Given a cost partition  $\mathbf{c} \in \mathbf{C}_p$ , computing  $h_{(\mathbf{c},\mathbf{u},\mathbf{b})}(s)$  for some abstraction  $(\mathbf{c}, \mathbf{u}, \mathbf{b}) \in H_{(\mathbf{c},\mathbf{u},-)}^p(s)$  is polynomial-time in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ , and  $\|\mathbf{u}\|$ .

*Proof:* The proof is by reduction to the polynomial fragment of the Knapsack problem corresponding to all items having identical value. Let  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$ , and, given that  $\mathbf{u}$  is a *strong* set of valued partitions, let  $\text{img}(\mathbf{u}[i]) = \{0, \sigma\}$ . For  $i \in [k]$ , let  $w_i$  be the cost of the cheapest path in  $\mathcal{T}_i$  from  $\alpha_i(s)$  to (one of the) states  $s' \in S_i$  with  $\mathbf{u}[i](s') = \sigma$ . Since  $\mathcal{AS}$  is an *explicit* abstraction skeleton, the set  $\{w_i\}_{i \in [k]}$  can be computed in time polynomial in  $\|\mathcal{AS}\|$  using one of the algorithms for the single-source shortest paths problem. Consider now a Knapsack  $\langle \{w_i, \sigma\}_{i \in [k]}, b \rangle$ , with weights  $w_i$  being as above and value  $\sigma$  being identical for all objects. Let  $X \subseteq [k]$  be a solution to that (optimization) Knapsack problem; recall that it is computable in polynomial time. Given that, we define budget profile  $\mathbf{b}^* \in \mathbf{B}$  as follows: for  $i \in [k]$ ,  $\mathbf{b}^*[i] = w_i$  if  $i \in X$ , and  $\mathbf{b}^*[i] = 0$ , otherwise.

What remains to be shown is that  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  actually induces an additive abstraction for  $\Pi$ , that is,  $\mathcal{A}_{(\mathbf{c},\mathbf{u},\mathbf{b}^*)} = \{\langle \mathcal{T}_i, \mathbf{c}[i], \mathbf{u}[i], \mathbf{b}^*[i] \rangle\}_{i \in [k]} \in_s \mathcal{AS}$ . Assume to the contrary that  $\mathcal{A}_{(\mathbf{c},\mathbf{u},\mathbf{b}^*)} \notin_s \mathcal{AS}$ , and let  $\pi$  be an optimal  $s$ -plan for  $\Pi$ . By the construction of our Knapsack problem and of  $\mathbf{b}^*$ , for each  $i \in X$ , there is a  $\alpha_i(s)$ -plan  $\pi_i$  in  $\langle \mathcal{T}_i, \mathbf{c}[i], \mathbf{u}[i], \mathbf{b}^*[i] \rangle$  with  $\hat{u}_i(\pi_i) = \sigma$ . According to Definition 1, our assumption implies that  $\hat{u}(\pi) > \sum_{i \in X} \hat{u}_i(\pi_i) = \sigma \cdot |X^*|$ . On the other hand, from Theorem 2, there exists a budget partition  $\mathbf{b} \in \mathbf{B}_p$  such that  $\mathcal{A}_{(\mathbf{c},\mathbf{u},\mathbf{b})} \in_s \mathcal{AS}$ . This budget partition induces a feasible solution  $X' = \{i \mid w_i \leq \mathbf{b}[i]\}$  for our Knapsack problem for which  $\hat{u}(\pi) \leq \sum_{i \in X'} \hat{u}_i(\pi_i) = \sigma \cdot |X'|$ . This, however, implies  $|X| < |X'|$ , contradicting our assumption, and thus accomplishing the proof of  $\mathcal{A}_{(\mathbf{c},\mathbf{u},\mathbf{b}^*)} \in_s \mathcal{AS}$ .  $\square$

The construction in the proof of Theorem 3 may ap-

task	60%			80%			100%		
	blind	basic	$h_{\mathcal{A}}$	blind	basic	$h_{\mathcal{A}}$	blind	basic	$h_{\mathcal{A}}$
blocks-4-0	23	23	23	47	37	36	19	19	19
blocks-4-1	10	10	10	30	26	24	13	13	13
blocks-4-2	13	13	13	25	22	19	11	11	11
blocks-5-0	53	27	13	191	104	47	20	20	20
blocks-5-1	86	55	35	281	174	54	75	61	46
blocks-5-2	58	37	37	316	231	163	176	161	138
blocks-6-0	124	116	77	440	380	225	34	34	34
blocks-6-1	658	300	79	2415	1401	125	489	358	134
blocks-6-2	394	213	59	3501	2463	266	3135	2800	1549
blocks-7-0	390	331	225	7387	4220	1411	8370	6382	3865
blocks-7-1	6604	3265	480	42168	28862	602	12012	9582	2069
blocks-7-2	3171	2632	1709	28632	20533	8179	15914	12922	8010
blocks-8-0	12691	6116	323	165980	100970	1869	130780	96403	4892
blocks-8-1	46723	27582	16670	347914	252303	—	36504	31174	21358
blocks-8-2	9535	3931	145	89450	46822	154	1035	846	367
blocks-9-1	8651	8099	5069	907991	598110	—	734526	519569	—
blocks-9-2	21488	8754	820	925192	518385	913	1128285	813209	9390
driverlog-1	47	47	27	80	80	48	36	36	36
driverlog-2	18500	18500	6035	93238	93238	40489	4307	4307	2126
driverlog-3	1039	1039	377	5649	5649	905	231	231	231
driverlog-4	31741	31741	1786	272699	272699	22308	292	292	292
driverlog-5	71224	71224	8255	1373724	—	—	1635025	—	—
driverlog-6	8477	8477	419	62817	62817	2015	8279	8279	3034
driverlog-7	31293	31293	1421	619572	—	14709	107312	107312	—
logistics-4-0	15532	15532	12487	70845	70845	42452	65601	65601	52339
logistics-4-1	10255	10255	8109	50217	50217	29402	29187	29187	22727
logistics-4-2	3766	3766	2260	14198	14198	8947	2808	2808	2808
logistics-5-0	69013	69013	40087	311846	311846	—	291620	291620	—
logistics-5-1	6410	6410	2473	23175	23175	10941	2082	2082	2082
logistics-5-2	154	154	119	653	653	262	57	57	57
logistics-6-0	47896	47896	21915	231351	231351	—	81987	81987	58819
logistics-6-1	2246	2246	711	9661	9661	3668	474	474	474
logistics-6-2	47032	47032	21400	228617	228617	—	89914	89914	64010
logistics-6-9	31536	31536	15521	162325	162325	66593	11574	11574	10026
depots-1	137	137	137	265	261	261	233	233	233
depots-2	1460	1404	1210	5887	4986	3269	1518	1518	1518

Table 1: Expanded node statistics for optimal OSP with BFBB search on a set of IPC tasks, cast as OSP.

pear somewhat counterintuitive: while we are interested in minimizing the heuristic estimate of  $h^*(s)$ , the abstraction  $\mathcal{A}_{(\mathbf{c},\mathbf{u},\mathbf{b}^*)}$  is selected via the value-maximizing Knapsack problem. However, a correct view of the situation would be that the selected triplet  $(\mathbf{c}, \mathbf{u}, \mathbf{b}^*)$  provides us with the lowest estimate of  $h^*(s)$  among all abstractions complying with  $\mathbf{u}$  and  $\mathbf{c}$ , whose  $\mathbf{A}(s)$  membership, aka admissibility, we know how to prove in polynomial time. Finally, while strong 0-binary value partitions are rather restrictive, finding an element of  $H_{(\mathbf{c},\mathbf{u},-)}^p(s)$  for general 0-binary  $\mathbf{u}$  is no longer polynomial—a reduction from Knapsack is straightforward. However, Knapsack is solvable in pseudo-polynomial time, and plugging that Knapsack algorithm into the proof of Theorem 3 results in a search algorithm for  $H_{(\mathbf{c},\mathbf{u},-)}^p(s)$  with general 0-binary  $\mathbf{u}$ , running in time polynomial (also) in the *unary representation* of the budget  $b$ .

For a first test of the value that additive abstractions can bring to heuristic-search OSP, we have prototyped a simple planning system on the basis of Pyperplan, a lightweight planner written in Python<sup>3</sup>. Within that prototype, we have provided support for some basic pattern-database abstraction skeletons, action cost partitions, and abstraction selection in  $H_{(\mathbf{c},\mathbf{u},-)}^p(s)$  for strong 0-binary value partitions as in the proof of Theorem 3. As best-first forward search algorithms such as  $A^*$  are not suitable for optimal OSP, we have implemented a best-first branch-and-bound (BFBB) search. This BFBB expands the nodes in the decreasing order of

<sup>3</sup><https://bitbucket.org/malte/pyperplan>

their state values, with the ties being broken towards higher  $h$ -values, and then higher remaining budgets. As our heuristic estimates always upper-bound the true values achievable from states, if the  $h$ -value of a generated state is lower than the best state value encountered so far, then that generated state is pruned. The search terminates when the search frontier becomes empty, and the optimal plan is then extracted from the search node associated with the best-value state encountered so far.

Table 1 compares BFBB node expansions with three heuristic functions, tagged *blind*, *basic*, and  $h_A$ , on set of IPC tasks that we cast as OSP by associating a separate value with each goal. With all three heuristics, the  $h$ -value of a node  $\sigma$  is set to 0 if the cost budget at  $\sigma$  is over-consumed. Otherwise, *blind* BFBB constitutes a trivial baseline in which  $h(\sigma)$  is simply set to the total value of all goals. In *basic* BFBB, each goal is associated with its atomic (that is, single variable) projection abstraction, and  $h(\sigma)$  is set to the total value of goals, each of which can be *individually* achieved within the respective projection abstraction (see Theorem 1), given the entire remaining budget. Finally,  $h_A$  is an additive abstraction heuristic that is selected from  $H_{(c,u,-)}$  as in the proof of Theorem 3, with  $c$  being an ad hoc cost partition over atomic projections of the planning task onto goal variables, and  $u$  being a value partition that associates the value of each goal (only) with the respective atomic projection.

Each task was approached under three different budgets, which were 60%, 80%, and 100% of the minimal cost needed to achieve all the goals in the task. Despite the simplicity of the abstraction skeletons, the number of nodes expanded by BFBB with  $h_A$  is typically substantially lower than the number of nodes expanded by *basic* BFBB, with the difference sometimes reaching three orders of magnitude.<sup>4</sup> While this evaluation is still very preliminary, it testifies to the practical prospects of additive abstractions for OSP.

Returning now to the algorithmic analysis in the context of strong 0-binary value partitions, we now proceed with relaxing the constraint of sticking to a fixed action cost partition  $c$ , thus buying more flexibility in selecting abstractions from  $H_{(-,u,-)}^p(s)$  (and improving the accuracy of our estimates), while still remaining computationally tractable.

**Definition 2** Let  $\Pi = \langle V, A; s, c, u, b \rangle$  be an OSP task,  $\mathcal{AS}$  be a homomorphic abstraction skeleton of  $\mathcal{T}(\Pi)$ , and  $u \in U_p$ . By  $\kappa_s(u)$  we refer to the largest value  $v \in \mathbb{R}^{0+}$  such that, for *each* action cost partition  $c \in C_p$ , there exists a budget partition  $b \in B_p$  with  $(c, u, b) \in H_{(-,u,-)}^p(s)$  and  $h_{(c,u,b)}(s) \geq v$ .

Note that  $\kappa_s(u)$  can be as low as 0 (and for us, “low” is good), even when, for any  $0 \leq v \leq \max_{s \in S} \sum_{i \in [k]} u[i](s)$ , there exists some  $(c, u, b) \in$

<sup>4</sup>The runtime inefficiency of Python turned out to be an unfortunate obstacle: within the allowance of 30 minutes, some instances were solved by *blind* BFBB and were not solved even by *basic* BFBB, and this despite an extremely simple computation of *basic*  $h$ -values.

$H_{(-,u,-)}^p(s)$  with  $h_{(c,u,b)}(s) \geq v$ . In particular, note that  $h(s) = \kappa_s(u)$  is at least as accurate as the estimate  $h_{(c,u,-)}(s)$  from Theorem 3 for *any* fixed cost partition  $c$ .

**Theorem 4 ( $H_{(-,u,-)}^p(s)$  & strong 0-binary  $u$ )**

Given an OSP task  $\Pi = \langle V, A; s, c, u, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS}$  of  $\mathcal{T}(\Pi)$ , and a strong 0-binary value partition  $u \in U_p$ , determining  $\kappa_s(u)$  is polynomial-time in  $\|\Pi\|$  and  $\|\mathcal{AS}\|$ .

Let  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$ , and, given that  $u$  is a strong 0-binary value partition, let  $\text{img}(u[i]) = \{0, \sigma\}$ . Note that, for each  $(c, u, b) \in H_{(-,u,-)}^p(s)$ , we have  $h_{(c,u,b)}(s) = m\sigma$  for some  $m \in \{0\} \cup [k]$ . Our algorithm for determining  $\kappa_s(u)$  is depicted in Figure 4, and its high-level flow is as simple as it gets: The for-loop of the algorithm decreasingly iterates over all the different estimates of  $h^*(s)$  that can possibly come from the abstractions in  $H_{(-,u,-)}^p(s)$ , testing whether  $\kappa_s(u)$  equals currently examined candidate estimate  $m\sigma$ . If the test (provided by the sub-routine *always-achievable*) is positive, then we are done. Otherwise, if the test fails for all  $m \in [k]$ , then  $\kappa_s(u) = 0$ , which in particular implies that no state with value greater than 0 can be reached from  $s$  in  $\Pi$  with budget  $b$ .

The test of *always-achievable* for  $\kappa_s(u) = m\sigma$  is based on a certain linear program  $\mathcal{L}_1(m)$ , the semantics of which is captured by Lemma 1 below.<sup>5</sup> Informally, if  $\mathcal{L}_1(m)$  is infeasible, then no cost partition over  $\mathcal{AS}$  can provide us with the additive estimate of  $m\sigma$ , and this *independently of the budget allowance*; this can happen only if all  $\sigma$ -valued abstract states are simply unreachable from the respective abstractions of  $s$  in at least  $k - m + 1$  tg-structures of  $\mathcal{AS}$ . Otherwise, if  $\mathcal{L}_1(m)$  is solvable, then its solution establishes an action cost partition over  $\mathcal{AS}$  that induces the *most costly achievement* of the additive estimate of  $m\sigma$  using  $\mathcal{AS}$ , with the respective total cost being captured in the solution by a specific LP variable  $\xi$ .

**Lemma 1** Given a planning task  $\Pi = \langle V, A; s, c, u, b \rangle$ , let  $\Pi/b' = \langle V, A; s, c, u, b' \rangle$ . For all  $m \in [k]$ , if  $x$  is a solution of  $\mathcal{L}_1(m)$ , then

$$x[\xi] = \max_{c \in C_p} \min \left[ b' \in B \mid \begin{array}{l} (c, u, b) \in H_{(-,u,-)}^p(s) \text{ w.r.t. } \Pi/b', \\ h_{(c,u,b)} \geq m\sigma \end{array} \right].$$

Otherwise, if  $\mathcal{L}_1(m)$  is infeasible, then for no  $\Pi/b'$  there exists  $(c, u, b) \in H_{(-,u,-)}^p(s)$  with respect to  $\Pi/b'$  such that  $h_{(c,u,b)} \geq m\sigma$ .

The correctness of the algorithm with respect to Theorem 4 stems from Lemma 1: Suppose that the algorithm terminates within the loop, and returns  $m\sigma$  for some  $m > 0$ . By the construction of the algorithm,  $\mathcal{L}_1(m)$  is feasible and if  $x$  is a solution of  $\mathcal{L}_1(m)$ , then  $x[\xi] \leq b$ . Lemma 1 then implies that, for *each* action cost partition  $c \in C_p$ , there exists a budget partition  $b \in B_p$  such that  $(c, u, b)$  is an additive abstraction for  $\Pi$  and  $h_{(c,u,b)}(s) \geq m\sigma$ . If  $m = k$ ,

<sup>5</sup>The proof of Lemma 1 is omitted for lack of space.



input:  $\Pi = \langle V, A; s, c, u, b \rangle$ ,  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$ ,  
 strong 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$   
 output:  $\kappa_s(\mathbf{u})$

**for**  $m = k$  **downto** 1 **do**  
   **if** always-achievable( $m$ ) **then return**  $m\sigma$   
**return** 0

always-achievable( $m$ ):  
   solve( $\mathcal{L}_1(m)$ )  $\mapsto$  *infeasible* / solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$   
   **if** *infeasible* **or**  $\mathbf{x}[\xi] > b$  **then return** false  
   **else return** true

solve( $\mathcal{L}_1(m)$ ):  
   set (5') to an arbitrary subset of constraints (5)  
   **loop**  
     set  $\mathcal{L}'_1(m)$  to  $\mathcal{L}_1(m)$ , with constraints (5') instead of (5)  
     ellipsoid-method( $\mathcal{L}'_1(m)$ )  $\mapsto$  *infeasible* / solution  $\mathbf{x} \in \text{dom}(\mathcal{X})$   
     **if** *infeasible* **return** *infeasible*  
     let  $\tau$  be a permutation of  $[k]$  such that  
        $\mathbf{x}[\mathbf{b}[\tau(1)]] \leq \mathbf{x}[\mathbf{b}[\tau(2)]] \leq \dots \leq \mathbf{x}[\mathbf{b}[\tau(k)]]$   
     **if**  $\mathbf{x}[\xi] \leq \sum_{i \in [m]} \mathbf{x}[\mathbf{b}[\tau(i)]]$  **then return**  $\mathbf{x}$   
     extend (5') with constraint  $\xi \leq \sum_{i \in [m]} \mathbf{b}[\tau(i)]$

Figure 4: An algorithm for computing  $\kappa_s(\mathbf{u})$  for strong 0-binary value partitions  $\mathbf{u} \in \mathbf{U}_p$  (Theorem 4).

then trivially  $\kappa_s(\mathbf{u}) = m\sigma$ . Otherwise, if  $m < k$ , we know that the algorithm did not terminate at the previous iteration corresponding to  $m + 1$ . Again, Lemma 1 implies that there exists an action cost partition  $\mathbf{c} \in \mathbf{C}_p$  for which no budget partition  $\mathbf{b}$  of  $b$  will induce an additive abstraction for  $\Pi$  with  $h_{(\mathbf{c}, \mathbf{u}, \mathbf{b})}(s) \geq (m + 1)\sigma$  (or, in case of infeasibility of  $\mathcal{L}_1(m + 1)$ , such a budget partition exists for no action cost partition at all.). Hence, by Definition 2,  $\kappa_s(\mathbf{u}) < (m + 1)\sigma$ , and in turn, by the structure of  $\mathbf{u}$ , that implies  $\kappa_s(\mathbf{u}) = m\sigma$ . Finally, if the algorithm terminates after the loop and returns 0, then precisely the same argument on the basis of Lemma 1 implies  $\kappa_s(\mathbf{u}) = 0$ .

It remains now to specify our linear programs  $\mathcal{L}_1(m)$  in detail, and analyze the complexity of solving them. Each such linear program is defined over variables

$$\mathcal{X} = \{\xi\} \cup \bigcup_{i \in [k]} \left[ \{d(s')\}_{s' \in \mathcal{T}_i} \cup \{\mathbf{b}[i]\} \cup \bigcup_{a \in A} \{\mathbf{c}[i](a)\} \right], \quad (2)$$

constraints as in Eqs. 3-5, and the objective of maximizing the value of  $\xi$ . The roles of the variables in  $\mathcal{L}_1(m)$  are as follows. Variable  $\mathbf{c}[i](a)$  captures the cost to be associated with label  $a$  in the tg-structure  $\mathcal{T}_i$ . For state  $s'$  in  $\mathcal{T}_i$ , variable  $d(s')$  captures the cost of the cheapest path in  $\mathcal{T}_i$  from  $\alpha_i(s)$  to  $s'$ , given that the transitions (aka edges) are weighted consistently with the values of the variables  $\mathbf{c}[i](\cdot)$ . Variable  $\mathbf{b}[i]$  captures the minimal budget needed for reaching in  $\mathcal{T}_i$  a state with value  $\sigma$  from state  $\alpha_i(s)$ , given that, again, the transitions are weighted consistently with the variable vector  $\mathbf{c}[i]$ . Finally,  $\xi$  captures the minimal total cost of reaching states with value  $\sigma$  in precisely  $m$  t-graphs induced by  $\mathcal{AS}$  under the joint performance measure  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$ .

The constraints of  $\mathcal{L}_1(m)$  are as follows. The first two

sets of constraints in (3) come from a simple LP formulation of the single source shortest paths problem with the source node  $\alpha_i(s)$ : optimizing  $\sum_{i \in [k]} \sum_{s' \in \mathcal{T}_i} d(s')$  under a fixed transition pricing  $\mathbf{c}$  leads to computing precisely that. The third set of constraints in (3) establishes the costs of the cheapest paths in  $\{\mathcal{T}_i\}$  from states  $\alpha_i(s)$  to states valued  $\sigma$ , enforcing the semantics of variables  $\mathbf{b}[i]$ . Constraints (4) are the cost partition constraints, enforcing  $\mathbf{c} \in \mathbf{C}_p$ . Finally, constraints (5) enforce the aforementioned semantics of the singleton variable  $\xi$ .

$$\begin{aligned} &\mathcal{L}_1(m) : \\ &\max \xi \quad \text{subject to} \\ &\quad \forall i \in [k] : \\ &\quad \left\{ \begin{array}{ll} d(s') = 0, & s' = \alpha_i(s) \\ d(s') \leq d(s'') + \mathbf{c}[i](a), & \forall (s'', a, s') \in \mathcal{T}_i \\ \mathbf{b}[i] \leq d(s'), & \forall s' \in \mathcal{T}_i, \mathbf{u}[i](s') = \sigma \end{array} \right. , \quad (3) \\ &\quad \forall a \in A : \sum_{i \in [k]} \mathbf{c}[i](a) \leq c(a), \quad (4) \\ &\quad \forall X \subseteq [k], |X| = m : \xi \leq \sum_{i \in X} \mathbf{b}[i]. \quad (5) \end{aligned}$$

Note that, while the number of variables, as well as the number of constraints in (3) and (4), are polynomial in  $|\Pi|$  and  $|\mathcal{AS}|$ , the number of constraints in (5) is  $\binom{k}{m}$ . Thus, solving  $\mathcal{L}_1(m)$  using standard methods for linear programming is not practical. However, using the ellipsoid algorithm for linear inequalities (Grotschel, Lovasz, and Schrijver 1981), an LP with an exponential number of constraints can be solved in polynomial time provided that an associated separation problem can be solved in polynomial time. In our case, the separation problem is, given an assignment to the variables of  $\mathcal{L}_1(m)$ , test whether it satisfied (3), (4), and (5), and if not, produce an inequality among (3), (4), and (5) violated by that assignment. We now show how our separation problem for  $\mathcal{L}_1(m)$  can be solved in polynomial time (see solve( $\mathcal{L}_1(m)$ ) in Figure 4) using what is called  $m$ -sum minimization LPs (Punnen 1992). As the number of constraints in (3) and (4) is polynomial, their satisfaction by an assignment  $\mathbf{x} \in \text{dom}(\mathcal{X})$  can be tested directly by substitution. For constraints (5), let  $\tau$  be a permutation of  $[k]$  such that  $\mathbf{x}[\mathbf{b}[\tau(1)]] \leq \mathbf{x}[\mathbf{b}[\tau(2)]] \leq \dots \leq \mathbf{x}[\mathbf{b}[\tau(k)]]$ . If  $\mathbf{x}[\xi] \leq \sum_{i \in [m]} \mathbf{x}[\mathbf{b}[\tau(i)]]$ , then it is easy to see that  $\mathbf{x}$  satisfies all the constraints in (5). Otherwise, we have our violated inequality  $\xi \leq \sum_{i \in [m]} \mathbf{b}[\tau(i)]$ .

### From Strong to General 0-Binary Value Partitions

Recall that the polynomial result of Theorem 3 easily extends to a pseudo-polynomial algorithm for general 0-binary value partitions. It turns out that a pseudo-polynomial extension of Theorem 4 is possible as well, though it is technically more involved.

#### Theorem 5 ( $H^p_{(-, \mathbf{u}, -)}(s)$ & 0-binary $\mathbf{u}$ )

Given an OSP task  $\Pi = \langle V, A; s, c, u, b \rangle$ , a homomorphic explicit abstraction skeleton  $\mathcal{AS}$  of  $\mathcal{T}(\Pi)$ , and a 0-binary

input:  $\Pi = \langle V, A; s, c, u, b \rangle$ ,  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$ ,  
 0-binary value partition  $\mathbf{u} \in \mathbf{U}_p$   
 output:  $\kappa_s(\mathbf{u})$

**let**  $0 < \epsilon < \min_{i \in [k]} \sigma_i$ ,  $\alpha = 0$ ,  $\beta = \sum_{i \in [k]} \sigma_i$   
**while**  $\beta - \alpha > \epsilon$  **do**  
    $v = \alpha + (\beta - \alpha) / 2$   
    $\text{solve}(\mathcal{L}_2(v)) \mapsto \text{infeasible} / \text{solution } \mathbf{x} \in \text{dom}(\mathcal{X})$   
   **if** *infeasible* **or**  $\mathbf{x}[\xi] > b$  **then**  $\beta = v$   
     **else**  $\alpha = v$   
**if**  $\alpha = 0$  **then return** 0; **else return**  $\beta$

$\text{solve}(\mathcal{L}_2(v))$ :  
 set (6') to an arbitrary subset of constraints (6)  
**loop**  
   set  $\mathcal{L}'_2(v)$  to  $\mathcal{L}_2(v)$ , with constraints (6') instead of (6)  
   ellipsoid-method( $\mathcal{L}'_2(v)$ )  $\mapsto \text{infeasible} / \text{solution } \mathbf{x} \in \text{dom}(\mathcal{X})$   
   **if** *infeasible* **return** *infeasible*  
   strict-Knapsack( $\{\mathbf{x}[\mathbf{b}[i]], \sigma_i\}_{i \in [k]}, \mathbf{x}[\xi]\}) \mapsto \text{solution } X \subseteq [k]$   
   **if**  $\sum_{i \in X} \sigma_i < v$  **then return**  $\mathbf{x}$   
   extend (6') with constraint  $\xi \leq \sum_{i \in X} \mathbf{b}[i]$

Figure 5: An algorithm for computing  $\kappa_s(\mathbf{u})$  for 0-binary value partitions  $\mathbf{u} \in \mathbf{U}_p$  (Theorem 5).

value partition  $\mathbf{u} \in \mathbf{U}_p$ , determining  $\kappa_s(\mathbf{u})$  within  $n$ -digit precision<sup>6</sup> is polynomial-time in  $\|\Pi\|$ ,  $\|\mathcal{AS}\|$ ,  $\log(n)$ , and a unary representation of the budget  $b$  of  $\Pi$ .

Let  $\mathcal{AS} = \{\mathcal{T}_i, \alpha_i\}_{i \in [k]}$ , and, for  $i \in [k]$ ,  $\text{img}(\mathbf{u}[i]) = \{0, \sigma_i\}$ . The algorithm for computing  $\kappa_s(\mathbf{u})$  for inputs as in Theorem 5 is depicted in Figure 5. The flow of that algorithm bears some similarity to the one in Figure 4, yet it is different in many respects.

At the high-level, the algorithm performs a binary search over the hypothesis interval  $[0, \sum_{i \in [k]} \sigma_i]$ . The parameter  $\epsilon$  serves as the “sufficient precision” criterion for termination; while any  $\epsilon > 0$  can be used, adopting  $\epsilon < \min_{i \in [k]} \sigma_i$  allows us to provide precision-independent answers in cases where  $\kappa_s(\mathbf{u}) = 0$ . At iteration corresponding to an interval  $[\alpha, \beta]$ , the algorithm attempts to solve a certain linear program  $\mathcal{L}_2(v)$ , testing the hypothesis  $\kappa_s(\mathbf{u}) \geq v$ , where  $v$  is the mid-point of  $[\alpha, \beta]$ . The test is positive if  $\mathcal{L}_2(v)$  is feasible and the value  $\mathbf{x}[\xi]$  in the respective solution  $\mathbf{x}$  for  $\mathcal{L}_2(v)$  indicates that, for the cost partition  $\mathbf{c}$  induced by  $\mathbf{x}$ , there is a budget partition  $\mathbf{b}$  that allows us to achieve the total (additive) estimate of at least  $v$  in  $t$ -graphs induced by  $\mathcal{AS}$  under the performance profile  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$ . If so, then the next hypothesis to test will be  $\kappa_s(\mathbf{u}) \geq v'$ , where  $v'$  is the midpoint of  $[v, \beta]$ . Otherwise, the next hypothesis corresponds to the midpoint of  $[\alpha, v]$ . To ensure admissibility of the estimate,

<sup>6</sup>The statement of Theorem 5 involves the precision of the estimate because the  $\sigma_i$  values of the abstract value functions  $\mathbf{u}[i]$  can be arbitrary real numbers. In the case of integer-valued sets of functions  $\mathbf{u}$ , as well as in various special cases of real-valued functions,  $\kappa_s(\mathbf{u})$  can be determined precisely using a simplification of the algorithm we introduce to support the claim of Theorem 5. These details, however, are more of a theoretical interest; for reasonably small values of  $\epsilon$ , in practice there will be no difference between estimates  $h(s)$  and  $h(s) + \epsilon$ .

upon termination of the loop, the estimate is set to  $\beta$ ; the only exception is the case of the last (unexamined) interval being  $[0, \epsilon]$ , in which the estimate is safely set to 0. The correctness of the algorithm with respect to Theorem 5 stems from a lemma on  $\mathcal{L}_2(v)$ , which is identical to Lemma 1, *mutatis mutandis*.

The LPs  $\mathcal{L}_2(v)$ ,  $v \in \mathbb{R}^{0+}$ , employed for testing hypotheses  $\kappa_s(\mathbf{u}) \geq v$ , are also defined over variables  $\mathcal{X}$  as in Eq. 2, and are obtained from  $\mathcal{L}_1(m)$  by replacing constraints (5) with constraints (6):

$$\mathcal{L}_2(v) : \max \xi \text{ subject to constraints (3), (4), and} \\ \forall X \subseteq [k] \text{ s.t. } \sum_{i \in X} \sigma_i \geq v : \quad \xi \leq \sum_{i \in X} \mathbf{b}[i]. \quad (6)$$

While the semantics of all variables but  $\xi$  remains as in  $\mathcal{L}_1(m)$ ,  $\xi$  now captures the minimal total cost of reaching some states  $\{s_i\}_{i \in [k]}$  from states  $\{\alpha_i(s)\}_{i \in [k]}$  in the respective  $k$   $t$ -graphs induced by  $\mathcal{AS}$  under the performance profile  $(\mathbf{c}, \mathbf{u}, \mathbf{b})$  such that  $\sum_{i \in [k]} \mathbf{u}[i](s_i) \geq v$ . The new constraints (6) enforce this semantics of  $\xi$  (and thus the required max-min semantics of  $\mathcal{L}_2(v)$ ). The number of constraints in (6) is  $\Theta(2^k)$ , and thus procedure  $\text{solve}(\mathcal{L}_2(v))$  also employs the ellipsoid method with a sub-routine for the associated separation problem. We now show how that separation problem for  $\mathcal{L}_2(v)$  can be solved in pseudo-polynomial time using a pseudo-polynomial procedure for the *strict* Knapsack problem. Given an assignment  $\mathbf{x} \in \text{dom}(\mathcal{X})$ , its feasibility with respect to (3) and (4) can be tested directly by substitution. For constraints (6), let  $X \subseteq [k]$  be an optimal solution to the strict Knapsack  $\langle \{\mathbf{x}[\mathbf{b}[i]], \sigma_i\}_{i \in [k]}, \mathbf{x}[\xi] \rangle$ .

- If the value  $\sum_{i \in X} \sigma_i$  of  $X$  is *smaller* than  $v$ , then  $\mathbf{x}$  satisfies *all* the constraints in (6). Assume to the contrary that  $\mathbf{x}$  violates some constraint in (6), corresponding to a set  $X' \subseteq [k]$ . By definition of (6),  $\sum_{i \in X'} \sigma_i \geq v$ , and by our assumption,  $\mathbf{x}[\xi] > \sum_{i \in X'} \mathbf{x}[\mathbf{b}[i]]$ . That, however, implies that  $X'$  is a feasible solution for our strict Knapsack, and of value higher than that of presumably optimal  $X$ .
- Otherwise, if  $\sum_{i \in X} \sigma_i \geq v$ , then  $X$  itself provides us with a constraint in (6) violated by  $\mathbf{x}$ .

## Summary

We defined and investigated fragments of additive abstractions for oversubscription planning. Along with revealing some significant islands of tractability, we exposed an interesting interplay between these abstractions and certain tools of combinatorial and convex optimization. Our empirical tests of the basic abstractions on a prototype system testified to the promise of the developed approach. Our next steps will thus be to develop an efficient implementation of the entire framework, and to engage in further formal investigation of what is hard and what is tractable in the context of devising quality abstractions for oversubscription planning.

**Acknowledgments.** This work was partially supported by the EOARD grant FA8655-12-1-2096, and the ISF grant 1045/12.

## References

- Bäckström, C., and Klein, I. 1991. Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence* 7(3):181–197.
- Bäckström, C., and Nebel, B. 1995. Complexity results for SAS<sup>+</sup> planning. *Computational Intelligence* 11(4):625–655.
- Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. A hybrid relaxed planning graph-LP heuristic for numeric planning domains. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *Proceedings of the International Conference on AI Planning and Scheduling (AIPS)*, 274–293.
- Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.
- Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning problems. *Journal of Artificial Intelligence Research* 20:61–124.
- Garey, M. R., and Johnson, D. S. 1978. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New-York: W.H. Freeman and Company.
- Grotschel, M.; Lovasz, L.; and Schrijver, A. 1981. The ellipsoid method and its consequences theorems in combinatorial optimization. *Combinatorica* 1:169–197.
- Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, 1007–1012.
- Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 200–207.
- Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*.
- Hoffmann, J. 2003. The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.
- Katz, M., and Domshlak, C. 2010a. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research* 39:51–126.
- Katz, M., and Domshlak, C. 2010b. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174:767–798.
- Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* 36:547–556.
- Punnen, A. P. 1992. K-sum linear programming. *The Journal of the Operational Research Society* 43(4):359–363.
- Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Pearson, 3 edition.

# On Combinatorial Actions and CMABs with Linear Side Information

Alexander Shleyfman and Antonín Komenda and Carmel Domshlak<sup>1</sup>

**Abstract.** Online planning algorithms are typically a tool of choice for dealing with sequential decision problems in combinatorial search spaces. Many such problems, however, also exhibit combinatorial actions, yet standard planning algorithms do not cope well with this type of “the curse of dimensionality.” Following a recently opened line of related work on combinatorial multi-armed bandit (CMAB) problems, we propose a novel CMAB planning scheme, as well as two specific instances of this scheme, dedicated to exploiting what is called linear side information. Using a representative strategy game as a benchmark, we show that the resulting algorithms very favorably compete with the state-of-the-art.

## 1 INTRODUCTION

In large-scale sequential decision problems, reasoning about the problem is often narrowed to a state space region that is considered most relevant to the specific decision problem currently faced by the agent. In particular, *online planning* algorithms focus only on the current state  $s_0$  of the agent, deliberate about the set of possible courses of action from  $s_0$  onwards, and, when interrupted, use the outcome of that exploratory deliberation to select an action to perform at  $s_0$ . Once that action is applied in the real environment, the planning process is repeated from the obtained state to select the next action and so on.

The basic components of any sequential decision problem are its states and actions. When the number of actions is polynomial in the size of the problem description, the basic computational complexity of planning stems solely from a prohibitive—exponential in the size of the problem representation—size of the state space. This “curse of state dimensionality” seems to receive most of the attention in the automated planning research. In particular, state-space forward search algorithms, including all standard, both systematic and Monte-Carlo, online planning algorithms, implicitly assume that the action choices at any state can be efficiently enumerated.

Whatever the atomic actions of the agents are, as long as the agent can perform only one (or a small fixed number of) atomic actions simultaneously, the above assumption of enumerable action choices is typically fine. However, if the agent we are planning for either represents a team of cooperating agents, or, equivalently, has a number of concurrent actuators, then the problem exhibits a “curse of action dimensionality” via the *combinatorial structure of the action space*.

Real-time strategy games (RTS) are a great example of decision problems with combinatorial action spaces, because a player is asked to activate *in parallel* a set of units that together form the force of the player [2, 6, 7, 15, 17]. That is, the set of actions  $A(s)$  available to a player at a state  $s$  corresponds to a (sometimes proper, due to some

game-specific constraints) subset of the cross-product of explicitly given sets of atomic actions of her units.

Previous work on online planning in RTS mostly avoided dealing with combinatorial actions directly, either by sequencing the decisions made for individual units [7, 15], or by abstracting the combinatorial action spaces to a manageable set of choices [2, 6]. The exception seems to be a recent work of Ontañón [16] that suggested considering combinatorial actions in online planning through the lens of *combinatorial multi-armed bandit (CMAB)* problems [11, 5, 16]. In particular, Ontañón suggested a specific Monte-Carlo algorithm for online planning in CMABs, called Naive Monte-Carlo (NMC), that is driven by an assumption that the expected value of a combinatorial action can be faithfully approximated by a linear function of its components. Evaluated on the  $\mu$ RTS game, NMC was shown to favorably compete with popular search algorithms such as UCT and alpha-beta ABCD, which avoid dealing with combinatorial actions directly [16].

Taking on board the CMAB perspective on combinatorial actions in sequential decision problems, here we continue the study of online planning algorithms for CMABs. In particular, we formalize the basic building blocks of such algorithms, as well as the trade-offs in computational resource allocation between them. Based on this analysis, we suggest a simple, two-phase scheme for CMAB planning, in which the first phase is dedicated solely to generating candidate combinatorial actions, and the second phase is dedicated solely to evaluating these candidates. Adopting the assumption of helpful linear “side information”, we then propose two instances of this two-phase scheme, LSI<sub>V</sub> and LSI<sub>F</sub>, that, in particular, build upon some recent developments around action selection in regular MAB problems [3, 4, 12]. Our experimental evaluation on the  $\mu$ RTS game as in [16] shows that both LSI<sub>V</sub> and LSI<sub>F</sub> substantially outperform NMC, as well as emphasizes the marginal value of both exploiting side information and of systematicity in candidate evaluation process.

## 2 BACKGROUND

The *multi-armed bandit (MAB)* problem is a sequential decision problem defined over a single state. At each stage, the agent has to execute one out of some  $k \geq 2$  stochastic actions  $\{a_1, \dots, a_k\}$ , with  $a_i$  being parameterized with an unknown distribution  $\nu_i$ , with expectation  $\mu_i$ . If  $a_i$  is executed, the agent gets a reward drawn at random from  $\nu_i$ .

Most research on MABs has been devoted to the setup of reinforcement learning-while-acting, where the performance of the agent is assessed in terms of its cumulative regret, the sum of differences between the expected reward of the best arm and the obtained rewards. Good algorithms for learning-while-acting in MAB,

<sup>1</sup> Technion, Israel, email: alesh@tx, akomenda@tx, dcarmel@ie.technion.ac.il

like UCB1 [1], trade off between exploration and exploitation. These MAB algorithms also gave rise to popular Monte-Carlo tree search algorithms for online planning in multi-state sequential decision problems (e.g., MDPs and sequential games), such as  $\varepsilon$ -MCTS [18], UCT [14], and MaxUCT [13].

However, as it was first studied in depth by Bubeck et al. [4], learning-while-acting and online planning are rather different problems that should favor different techniques. Unlike in learning-while-acting, the agent in online planning may try the actions “free of charge” a given number of times  $N$  (not necessarily known in advance) and is then asked to output a recommended arm. The agent in online planning is evaluated by his *simple regret*, i.e., the difference  $\mu^* - \mu_i$  between the expected payoff of the best action and the average payoff obtained by his recommendation  $a_i$ . In other words, the rewards obtained by the agents at planning are fictitious. Therefore, good algorithms for online planning in MABs, like uniform-EBA [4], Successive Rejects [3], and SequentialHalving [12], are focused solely on exploration, and they already gave rise to efficient Monte-Carlo tree search algorithms for online planning in multi-state sequential decision problems such as BRUE [9] and MaxBRUE [10].

In contrast to regular MAB problems, in which rewards are associated with individual actions and a single action is executed at each stage, in *combinatorial multi-armed bandit (CMAB) problems*, the rewards are associated with certain subsets of actions, and the agent is allowed to simultaneously execute such subsets of actions at each stage [11, 5, 16]. In terms closest to problems that motivated our work in the first place, i.e., sequential decision problems for teams of cooperative agents, a CMAB problem is given by a finite set of  $n \geq 1$  classes of actions  $\{A_1, \dots, A_n\}$ , with  $A_i = \{a_{i,1}, \dots, a_{i,k_i}\}$ , and a constraint  $C \subseteq \mathcal{A} = [A_1 \cup \{\epsilon\}] \times \dots \times [A_n \cup \{\epsilon\}]$ , where  $\epsilon$  denotes “do nothing”, and thus  $\mathcal{A}$  is the set of all possible subsets of actions, with at most one representative from each action class. We refer to every set of actions  $\mathbf{a} \in \mathcal{A}$  as a combinatorial action, or *c-action*, for short. Each c-action  $\mathbf{a}$  is parameterized with an unknown distribution  $\nu(\mathbf{a})$ , with expectation  $\mu(\mathbf{a})$ . At each stage, the agent has to execute one out of some  $2 \leq K = |\mathcal{C}| \leq \prod_{i=1}^n k_i$  c-actions, and if c-action  $\mathbf{a}$  is executed, then the agent gets a reward drawn at random from  $\nu(\mathbf{a})$ .

Whether our setup is online planning in CMABs or learning-while-planning in CMABs, it is easy to see that CMAB problems with  $K = O(\text{poly}(n))$  can be efficiently approached with regular MAB algorithms. However, if the problem is only loosely constrained and thus the c-action space grows exponentially with  $n$  (as it is typically the case in RTS-like planning problems), then the algorithms for regular MAB problems are no-go because they all rely on assumption that each c-action can be sampled at least once. This led to devising algorithms for CMAB learning-while-planning [11, 5] and online planning [16], all making certain assumptions of “side information”, usefulness of which depends (either formally or informally) on the properties of  $\mu$  over the polytope induced by  $A_1 \times \dots \times A_n$ . Such a “side information” basically captures the structure of  $\mu$  targeted by the algorithm, but the algorithm can still be sound for arbitrary expected reward functions. This is, for instance, the case with the Naive Monte-Carlo algorithm of Onta  n [16], which we describe in detail and compare to, later on.

### 3 ONLINE PLANNING IN CMABs

Due to the “curse of action space dimensionality”, at a high level, any algorithm for online planning in CMABs should define two strategies:

1. a *candidate generation strategy*, for reducing the set of candidates from  $\mathcal{C}$  to a reasonably small subset  $\mathcal{C}^* \subseteq \mathcal{C}$  of candidates, and
2. a *candidate evaluation strategy*, for identifying the best c-action in  $\mathcal{C}^*$  by gradually improving the corresponding estimates of  $\mu$ .

Given such a pair of strategies, the overall algorithm can then apply them, either sequentially or in interleaving, to sample the selected c-actions. The question is, of course, what pair of strategies to adopt, and how to combine between them so to best exploit the available planning time. The only previous proposal in that respect corresponds to the recent Naive Monte-Carlo (NMC) algorithm of Onta  n [16]. At a high level, NMC constitutes a composition of  $\varepsilon$ -greedy sampling strategies, operated under an assumption that  $\mu$  is linear in the atomic actions of the CMAB, i.e.,  $\mu(\mathbf{a}) = \sum_{i=1}^n \sum_{j=1}^{k_i} 1_{\{a_{i,j} \in \mathbf{a}\}} w_{i,j}$ , where  $1_{\{\cdot\}}$  is the indicator function, and  $w_{i,j} \in \mathbb{R}$ . Specifically, at each stage, NMC follows the candidate generation/evaluation strategy with probability  $\varepsilon_0/(1 - \varepsilon_0)$ , respectively, where:

1. The candidate generation strategy generates and samples a candidate c-action  $\mathbf{a}$  by selecting atomic actions from each set  $A_i$  independently and  $\varepsilon$ -greedily: with probability  $\varepsilon_1$ ,  $\mathbf{a}$  will contain the “empirically best atomic action” from  $A_i$ , and with probability  $(1 - \varepsilon_1)$ , the  $i$ -th component of  $\mathbf{a}$  will be selected from  $A_i$  uniformly at random. Atomic action  $a_{i,j}$  is “empirically best” in  $A_i$  if, so far, the average reward of the (c-action) samples involving  $a_{i,j}$  is the highest among the elements of  $A_i$ .
2. The candidate evaluation strategy samples the empirically best action in (the current) set  $\mathcal{C}^*$ .

At the end, the algorithm output (and the agent performs) the best empirical action in  $\mathcal{C}^*$ .

Assuming  $\varepsilon_0, \varepsilon_1 < 1$ , every c-action in  $\mathcal{C}$  will eventually be generated and then sampled infinitely often. Thus, NMC converges in the limit to the best c-action in  $\mathcal{C}$ , and this independently of whether the assumption of  $\mu$ ’s linearity in atomic actions actually holds. Moreover, in an empirical evaluation on  $\mu$ RTS game in [16], NMC was shown to substantially outperform the standard tree search algorithms, such as UCT and ABCD, showing the promise of CMAB planning algorithms in decision problems with combinatorial actions. This precisely was the departing point for our work here.

Considering the dynamics of NMC, we note that candidate generation and candidate evaluation in it are stochastically interleaved, and the interleaving is made at the resolution of single samples. One possible motivation for such an interleaving might be in exploiting samples made at the evaluation samples to improve the estimated side information (aka linear function coefficients) for the generation steps. However, a closer look suggests that such an interleaving of candidate generation and candidate evaluation disadvantages the planning process twofold.

- If  $m$  samples are getting devoted directly to candidate generation, then the algorithm will generate up to  $m$  new candidates. Thus, even if the side information assumptions hold, a vast majority of the candidates will unavoidably be generated without a quality guidance of this side information as the latter is being acquired gradually over the candidate generation process.
- More importantly, while NMC converges to the best c-action in the limit, for no reasonable budget of samples  $N = o(K)$  it can provide any meaningful guarantees on the quality of the recommended action, not only with respect to the entire set of choices  $\mathcal{C}$  (which is understandable), but even *with respect to the generated subset of candidates  $\mathcal{C}^*$* .

The latter issue appears to be especially concerning, and, in particular, it stems from the fact that, after any number  $N = o(K)$  of samples, the best empirical mean among the c-action in  $C^*$  might be based on just a single sample of the respective c-action.

Taking that on board, in what follows we examine the prospects of algorithms that exhibit no interleaving of candidate generation and candidate evaluation at all. These algorithms take a simple two-phase approach of dividing the overall sample allowance  $N$  between the candidate generation phase that *runs first*, using  $N_g$  samples, and the candidate evaluation phase that *runs second*, using  $N_e = N - N_g$  samples. The motivation behind this simple two-phase scheme is twofold. Fixing some  $k$  c-action candidates  $C^*$  induces a problem of online planning in regular MAB, and state-of-the-art algorithms for this problem guarantee that the probability of choosing sub-optimal c-action from  $C^*$  decreases *exponentially* with  $N_e$  [4, 3, 12]. Reversely, suppose that, given a sample allowance  $N_e$  for the candidate evaluation phase, the algorithm of our choice for this phase guarantees that the recommended c-action will indeed be the best among  $C^*$  with probability of at least  $\delta(k)$ . If we are interested in choice-error probability of at most  $\delta$ , then there is no point in coming up with more than some  $k(\delta, N_e)$  candidate c-actions, and thus the candidate generation phase can/should be optimized to selection of precisely that number of candidates.

In what follows, we suggest and evaluate two simple variants of two-phase online planning for CMAB, LSI<sub>V</sub> (short for “linear side information from vertices”) and LSI<sub>F</sub> (short for “linear side information from facets”). Both algorithms assume the same type of helpful side information, namely that  $\mu$  is faithfully approximated by a function that is linear in the atomic actions of the CMAB, and differ only in the way this side information is actually estimated. Some auxiliary notation:  $\llbracket n \rrbracket$  for  $n \in \mathbb{N}$  denotes the set  $\{1, \dots, n\}$ . For a finite-domain, non-negative, real-valued function  $f$ ,  $\mathcal{D}[f]$  denotes a probability distribution over the domain of  $f$ , obtained by normalizing  $f$  as a probability function using a normalization of our choice. For such a probability distribution  $\mathcal{D}[f]$  and a non-empty subset  $S$  of the  $f$ ’s domain, by  $\mathcal{D}[f \upharpoonright_S]$  we refer to the conditional of  $\mathcal{D}[f]$  on  $S$ . Finally, the operation of drawing a sample from a distribution  $\mathcal{D}$  is denoted by  $\sim \mathcal{D}$ .

Figure 1a depicts the two-phase sampling scheme underlying both LSI<sub>V</sub> and LSI<sub>F</sub>. Given a partition of sample budget  $N$  into  $N_g$  and  $N_e$ , the algorithms first generates  $k(N_e)$  c-actions (GENERATE), and then evaluates these c-actions to recommend one of them (EVALUATE). The GENERATE procedure comprises

- (1) generating a weight function  $\hat{R}$  from atomic actions (adopting the linear side information assumption);
- (2) schematically generating a probability distribution  $\mathcal{D}_{\hat{R}}$  over c-action space  $\mathcal{C}$ , biased “towards”  $\hat{R}$ ; and
- (3) sampling (up to)  $k(N_e)$  c-actions  $\mathcal{C}$  from  $\mathcal{D}_{\hat{R}}$ .

EVALUATE then implements the recent *SequentialHalving* algorithm of Karnin et al. [12] for action recommendation (aka online planning) in regular MABs. Any other algorithm for this problem will do as well, but *SequentialHalving* provides the best formal guarantees to date, and it is the algorithm we have used in our empirical evaluation discussed later on.

Steps (1) and (2) of GENERATE are formulated above at high level, and there is a number of ways one can implement these steps. Considering step (1), if  $\mu$  is indeed linear in atomic actions and  $\mathcal{C}$  comprises all the possible combinations of atomic actions, then one can simply (i) pick an arbitrary set of  $|A|$  c-actions that span the atomic actions  $A$ , (ii) use the average rewards obtained from sampling these

actions equally often to construct a linear  $|A| \times |A|$  system, (iii) solve this system to obtain the coefficients  $w_{i,j}$  of  $\mu$ , and (iv) skip the EVALUATE step, recommending the c-action that maximizes  $\mu$ . However, both  $\mu$  can be very much non-linear, and the constraint  $\mathcal{C}$  can be arbitrary complex. Thus, the side information should be estimated and used in a way that relies on, yet is not constrained by, the side information assumption.

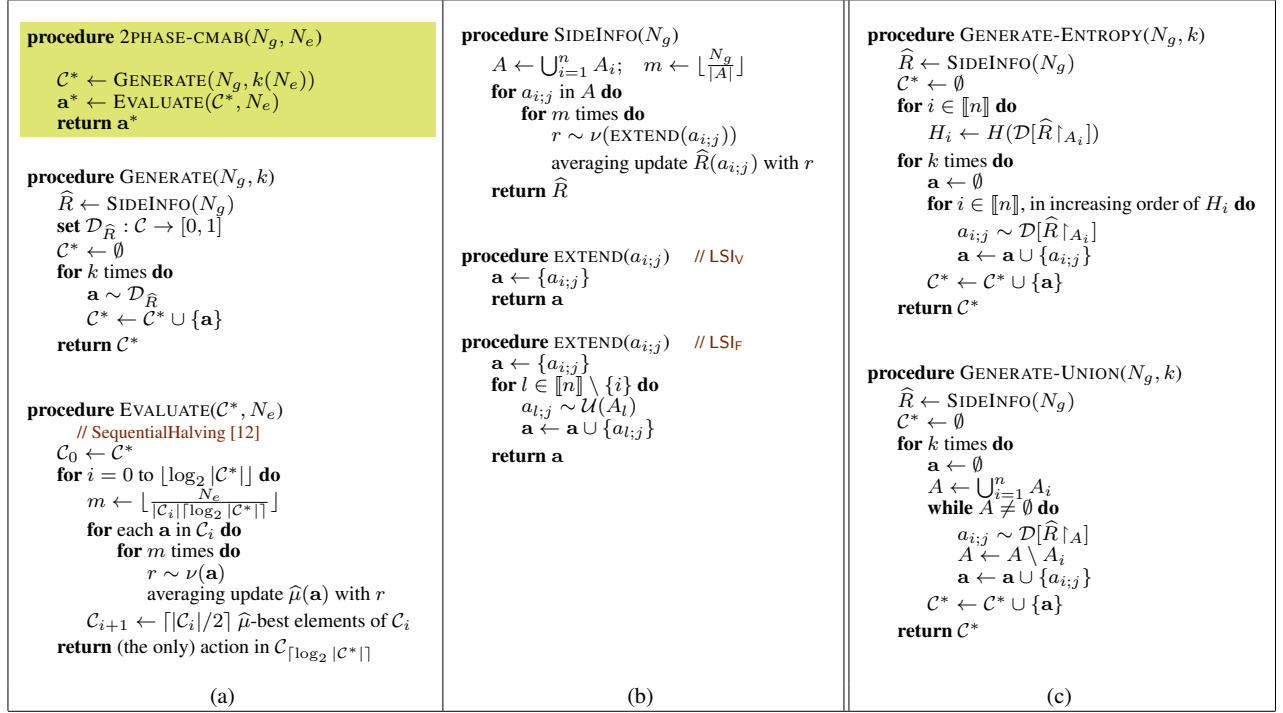
Given that, in SIDEINFO, both algorithms partition the sample allowance  $N_g$  equally between the atomic actions, and, for each atomic action  $a_{i,j}$ , set its weight  $\hat{R}(a_{i,j})$  to the average reward obtained from sampling some c-actions containing  $a_{i,j}$ . This is precisely the point where LSI<sub>V</sub> and LSI<sub>F</sub> slightly differ, and Figure 1b depicts the two corresponding versions of the EXTEND subroutine.

- In LSI<sub>V</sub>, all the  $m$  samples in  $a_{i,j}$ ’s budget are dedicated to a single c-action, notably the c-action comprising only  $a_{i,j}$ . (In RTS, this corresponds to a c-action that activates unit  $i$  while leaving all other units idle.)
- In LSI<sub>F</sub>, the  $m$  samples in  $a_{i,j}$ ’s budget go to some  $\leq m$  c-actions containing  $a_{i,j}$  that are generated uniformly at random.

In other words, LSI<sub>V</sub> establishes weights  $\hat{R}$  by sampling all the  $\sum_{i=1}^n k_i$  neighbors of a single vertex of the polytope induced by  $\mathcal{A}$ , and LSI<sub>F</sub> establishes weights  $\hat{R}$  by sampling the  $\alpha$  facets induced by the atomic actions  $A$  on that polytope. A priori, the relative attractiveness of LSI<sub>V</sub> and LSI<sub>F</sub> can be assessed only heuristically: The closer  $\mu$  is to satisfy the assumption of linearity, the more advantageous LSI<sub>V</sub> seems to be, and the other way around, the farther  $\mu$  is from linearity, the more relatively reliable is the side information provided by LSI<sub>F</sub>.

Proceeding now with step (2) of GENERATE, i.e., using the weight function  $\hat{R}$  to fix a probability distribution  $\mathcal{D}_{\hat{R}}$  over  $\mathcal{C}$ , in Figure 1c we show two specific realizations of this step, GENERATE-ENTROPY and GENERATE-UNION. Both these realizations are motivated by the fact that  $\mathcal{C}$  can comprise an arbitrary subset of the entire cross-product of the atomic action classes, and in both,  $\mathcal{D}_{\hat{R}}$  is specified implicitly, via auxiliary distributions over subsets of atomic actions, and step (2) is effectively combined with step (3) of sampling  $k(N_e)$  c-actions  $\mathcal{C}$  from  $\mathcal{D}_{\hat{R}}$ .

- In GENERATE-ENTROPY, the atomic action classes are ordered in the increasing order of entropy that is exhibited by the corresponding probability distributions  $\mathcal{D}[\{\hat{R}(a_{i;1}), \dots, \hat{R}(a_{i;k_i})\}]$ , as measured by an entropy measure  $H$  (such as the Shannon entropy, or some other Renyi entropy [8]). These measures quantify the diversity of probability distributions, and minimize on the least diverse distributions, which are uniform distributions. Hence, if c-actions are generated by sampling the atomic action classes sequentially, yet these sequential choices are inter-constrained, sampling the action classes in the increasing order of  $H(\mathcal{D}[\{\hat{R}(a_{i;1}), \dots, \hat{R}(a_{i;k_i})\}])$  prioritizes classes in which the different atomic actions actually differ in their purported value, and thus the choice really matters.
- In GENERATE-UNION, the action classes are not sampled independently, but each c-action added to  $\mathcal{C}$  is generated by sampling the union of all the atomic actions according to  $\mathcal{D}[\hat{R} \upharpoonright_A]$ , iteratively updating the conditional  $A$  to contain only actions from classes that are yet to be represented in the constructed c-action candidate.



**Figure 1.** (a) The general 2PHASE-CMAB planning scheme, as well as the LSI scheme for candidate generation and evaluation, (b) specifics of the LSI<sub>V</sub> and LSI<sub>F</sub> instances of 2PHASE-CMAB, and (c) two specific procedures for LSI candidate generation.

## 4 TWO-PHASE CMAB MEETS RTS

In what follows, we report on an empirical evaluation of 2PHASE-CMAB on top of the  $\mu$ RTS game platform of Onta  n [16].<sup>2</sup> Our objective in this evaluation was

- to examine the relative effectiveness of 2PHASE-CMAB in general, and of LSI<sub>V</sub> and LSI<sub>F</sub> in particular,
- to examine the marginal contribution of the two-phases of 2PHASE-CMAB, and
- to examine the relevance of CMAB planning to multi-state sequential planning problems with combinatorial actions, such as RTS games.

The  $\mu$ RTS platform already contained an implementation of Naive Monte-Carlo (NMC), with parameters optimized as in [16], and we have added an implementation of LSI<sub>V</sub> and LSI<sub>F</sub>; henceforth, superscripts  $e$  and  $u$  denote the versions of these algorithms using GENERATE-ENTROPY and GENERATE-UNION, respectively.

### 4.1 $\mu$ RTS

$\mu$ RTS is a two-player zero-sum game that exhibits standard features of popular RTS games, and in particular, heterogeneous units with durative actions that can be activated concurrently. In our experiments we use the  $8 \times 8$  grid environment. The environment is fully observable, and each grid cell can be occupied either by a single unit, or by a building, or by a resource storage. The storages each have a

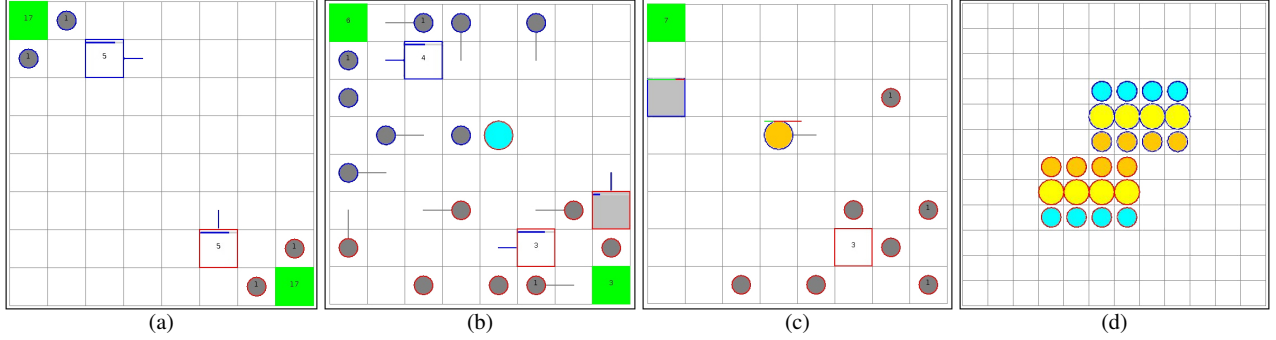
limited supply of the resource, and they can be used by both players. Table 1 shows the parameters of units and buildings. A player can build working units and combat units. The working units are all identical (*Worker*), and they can move the resources around, build buildings, and attack other units. As attackers, however, they are weak. The combat units come in three types—light melee (*LMelee*), heavy melee (*HMelee*), and ranged unit (*Ranged*)—all better attackers than working units, each with its own strengths and weaknesses. In general, movements and attacks are possible only within the 4-neighbourhood of the unit; all actions are durative and not interruptible. Finally, working units and combat units can be built only in *Base* and *Barracks* buildings, respectively.

	HP	Cost	$T(\text{Move})$	Damage	Range	$T(\text{Prod})$
Base	10	10	—	—	—	250
Barracks	4	5	—	—	—	200
Worker	1	1	10	1	1	50
LMelee	4	2	8	2	1	80
HMelee	4	2	12	4	1	120
Ranged	1	2	12	2	3	100

**Table 1.** Parameters of different buildings and units in  $\mu$ RTS. *HP* stands for health points, *Cost* is in the resource units,  $T(\text{Move})$  is the duration of a single move (in simulated time units), *Damage* and *Range* represent decrease of *HP* of the target unit and the range of the attack, respectively, and  $T(\text{Prod})$  is the duration of producing the unit/building.

For each player, the initial state of the game contains one *Base*, one *Worker* near the base, and one nearby resource storage, which, even if the resources are gathered optimally, suffices only for 1/3 of the maximal game duration. The game is restricted to 3000 simulated

<sup>2</sup> We would like to thank Santiago Onta  n for making the  $\mu$ RTS platform available to the public.



**Figure 2.** Three typical phases of the game in terms of unit counts: (a) early, (b) mid, and (c) end phases. Picture (d) depicts a “face-up” complex decision scenario: top to bottom, the rows of units are *Ranged*, *HMelee*, and *LMelee* of one player, and then, in reverse order, the units of the other player.

time units, and typically, it evolves in three phases in terms of the number of controlled units (and thus, the number of alternative c-actions): (i) early game (less units), (ii) mid game (more units) and (iii) end game (less units). Figures 2 depict representative states from these three phases.

## 4.2 Experiments

In our experiments, we compared the performance of  $LSI_V$  and  $LSI_F$  to that of NMC. In  $\mu$ RTS settings, the latter was already shown to substantially outperform both state-of-the-art tree search algorithms, such as ABCD,  $\epsilon$ -greedy, and UCT, as well as regular MAB algorithms and some handcrafted heuristics [16]. As a baseline, we also added two basic algorithms that were already implemented in  $\mu$ RTS, namely

Random, selecting a random action for each agent as soon as it can act, and

LRush, a handcrafted heuristic policy, corresponding to, first, optimally gathers resources with one of the workers and building a *Barracks* as soon as it becomes possible, and after that, building only *LMelee* units which go towards and attack the closest enemy units and buildings.

Considering 2PHASE-CMAB, we have implemented  $LSI_V$  and  $LSI_F$  with both GENERATE-ENTROPY and GENERATE-UNION, resulting in four 2PHASE-CMAB algorithms:  $LSI_V^e$ ,  $LSI_V^u$ ,  $LSI_F^e$ , and  $LSI_F^u$ . In line with the  $\epsilon_0$  parameter of NMC being preset to 0.25, we have set the  $N_g$  and  $N_e$  parameters of 2PHASE-CMAB to  $0.25N$  and  $0.75N$ , respectively. The number of candidates  $k(N_e)$  was set so that the first iteration of SequentialHalving will sample each candidate at least once. The  $H$ -measure in GENERATE-ENTROPY was set to the Shannon entropy.

Likewise, to assess the marginal value of exploiting side information, we have also implemented a simplified instance, noSI, that selects  $k(N_e)$  c-action candidates uniformly at random, and then passes these candidates to EVALUATE that implements SequentialHalving like  $LSI_V$  and  $LSI_F$ . In other words, noSI is a purified version of 2PHASE-CMAB that relies on no side information whatsoever. Importantly, to allow a meaningful comparison, the number of candidates  $k(N_e)$  in noSI was set exactly as in  $LSI_V/LSI_F$  variants, yet the trivial GENERATE phase of noSI then uses only  $k(N_e) \ll N_g$  samples, throwing out the residual  $N_g - k(N_e)$  samples.

To reduce as much as possible the effect of the variance, as well as of possible biases of the game simulator, each algorithm played

against each other algorithm 600 games, 300 games as “player 1” and 300 games as “player 2”. For all the CMAB algorithms (including NMC),

- the overall computational effort was set to  $N = 2000$  samples per decision;
- each sample comprised a simulated rollout of 200 game time units, with actions along the rollouts being selected at random, with a bias towards towards attacking a unit in reach, if there is such; and
- for rollouts ending at non-terminal states, the reward was assessed by a build-in evaluation function, reflecting the number of units and their health at the rollout’s end-state.

This lookahead and evaluation procedure is similar to the one used in the original evaluation of NMC [16].

Table 2 shows the results of this head-to-head competition between the algorithms.

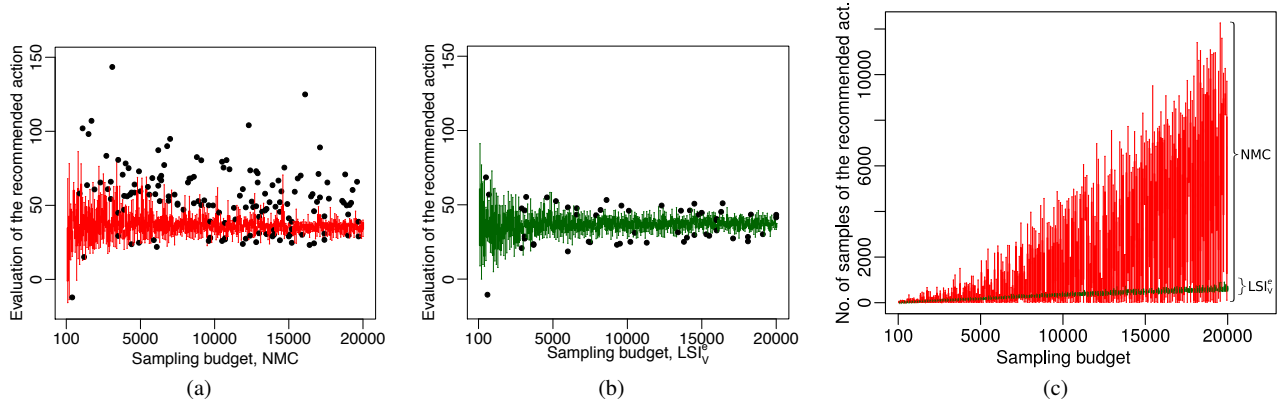
- Consistently with the previous experiments of Onta  n [16], all the CMAB algorithms easily defeated both Random and LRush, with the LSI instances never losing to these two baselines. The latter is not so for NMC, but its outperforming of Random and LRush is also a clear cut.
- All the four LSI instances of 2PHASE-CMAB consistently outperformed noSI. At the same time, the performance of the LSI instances among themselves was rather on par. In sum, this performance of LSI vs. noSI testifies both for the usefulness of linear side information (at least in this specific benchmark), as well as for the ability of the four GENERATE procedures of LSI instances to home in on this side information.
- All the five 2PHASE-CMAB instances, including noSI, substantially outperformed NMC. These results, and especially the result for noSI vs. NMC, strongly testify for the importance of a systematic candidate evaluation, and a controlled choice of the number of candidates to evaluate.

The latter point is even stronger supported by the results of an additional experiment that we performed on a complex decision scenario with 12 combat units at each side (see Figure 2d), and 5184 applicable c-actions per player. Figures 3a and 3b show that the empirical mean and variance of the value that NMC and  $LSI_V^e$  estimated for the c-actions they ended up recommending were rather similar, and this consistently over different sizes of sample budget. At the same time, the number and the magnitude of the outliers, especially of those overestimating the evaluation, were substantially larger with NMC. These overestimates are critical in strategic scenarios as the



w/t/l $\rightarrow$	Random	LRush	NMC	noSI	LSI <sub>V</sub> <sup>e</sup>	LSI <sub>V</sub> <sup>u</sup>	LSI <sub>F</sub> <sup>e</sup>	LSI <sub>F</sub> <sup>u</sup>
Random	38/27/35	94/0/6	100/0/0	100/0/0	100/0/0	100/0/0	100/0/0	100/0/0
LRush		0/100/0	96/0/4	98/0/2	100/0/0	100/0/0	100/0/0	100/0/0
NMC			41/13/46	52/12/36	54/14/32	55/14/31	54/15/31	52/15/32
noSI				42/17/41	46/17/40	47/14/38	44/17/39	46/17/37
LSI <sub>V</sub> <sup>e</sup>					40/18/42	42/19/40	42/16/42	43/18/39
LSI <sub>V</sub> <sup>u</sup>						41/17/42	39/16/44	45/15/40
LSI <sub>F</sub> <sup>e</sup>							43/17/40	41/17/42
LSI <sub>F</sub> <sup>u</sup>								41/16/43

**Table 2.** The results of the head-to-head competition: the percentage of wins/ties/looses of the column algorithm against the row algorithm.



**Figure 3.** Empirical mean and variance of the estimated value of the c-action recommended by NMC (a) and LSI<sub>V</sub><sup>e</sup> (b) in a complex “face-up” decision scenario (Figure 2d), after different sampling budgets (x-axis), ten runs per sample budget. The dots show the outliers that deviate three standard deviation from the mean. Respectively, (c) depicts the variance in the number of samples dedicated to the recommended c-action by NMC and LSI<sub>V</sub><sup>e</sup>.

“face-up” scenario used in the experiments, because making a particularly bad decision here can determine loosing the entire game. An example of a such decision in  $\mu$ RTS is whether to build *Barracks* or not at an early stage of the game. As it is illustrated in Figure 3c, these drastically overestimating outliers are caused by the extreme variance in the number of samples used by NMC to estimate the value of the c-action that ends up being recommended, with quite often the recommendation being based on just a single estimating sample of the respective c-action. On the contrary, the variance in the number of samples used to estimate the value of the recommended c-action in the systematic candidate evaluation procedure of LSI<sub>V</sub><sup>e</sup> is almost negligible, making the c-action selection process much more robust.

#### ACKNOWLEDGMENTS

This work was partly supported by USAF EOARD (grant no. FA8655-12-1-2096), the Technion- Microsoft Electronic-Commerce Research Center, and a Technion fellowship.

#### References

- [1] P. Auer, N. Cesa-Bianchi, and P. Fischer, ‘Finite-time analysis of the multiarmed bandit problem’, *Machine Learning*, **47**(2-3), 235–256, (2002).
- [2] R. Balla and A. Fern, ‘UCT for tactical assault planning in real-time strategy games’, in *IJCAI*, pp. 40–45, (2009).
- [3] S. Bubeck and R. Munos, ‘Open loop optimistic planning’, in *COLT*, pp. 477–489, (2010).
- [4] S. Bubeck, R. Munos, and G. Stoltz, ‘Pure exploration in finitely-armed and continuous-armed bandits’, *Theor. Comput. Sci.*, **412**(19), 1832–1852, (2011).
- [5] W. Chen, Y. Wang, and Y. Yuan, ‘Combinatorial multi-armed bandit: General framework and applications’, in *ICML*, pp. 151–159, (2013).
- [6] M. Chung, M. Buro, and J. Schaeffer, ‘Monte Carlo planning in RTS games’, in *IEEE-CIG*, (2005).
- [7] D. Churchill, A. Saffidine, and M. Buro, ‘Fast heuristic search for RTS game combat scenarios’, in *AIIDE*, (2012).
- [8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, 2 edn., 2006.
- [9] Z. Feldman and C. Domshlak, ‘Monte-Carlo planning: Theoretically fast convergence meets practical efficiency’, in *UAI*, (2013).
- [10] Z. Feldman and C. Domshlak, ‘On MABs and separation of concerns in Monte-Carlo planning for MDPs’, in *ICAPS*, (2014).
- [11] Y. Gai, B. Krishnamachari, and R. Jain, ‘Learning multiuser channel allocations in cognitive radio networks: A combinatorial multi-armed bandit formulation’, in *IEEE Symposium on New Frontiers in Dynamic Spectrum*, pp. 1–9, (2010).
- [12] Z. S. Karnin, T. Koren, and O. Somekh, ‘Almost optimal exploration in multi-armed bandits’, in *ICML*, pp. 1238–1246, (2013).
- [13] T. Keller and M. Helmert, ‘Trial-based heuristic tree search for finite horizon MDPs’, in *ICAPS*, pp. 135–143, (2013).
- [14] L. Kocsis and C. Szepesvári, ‘Bandit based Monte-Carlo planning’, in *ECML*, pp. 282–293, (2006).
- [15] A. Kovarsky and M. Buro, ‘Heuristic search applied to abstract combat games’, in *Canadian Conference on AI*, volume 3501 of *LNCIS*, pp. 66–78, (2005).
- [16] S. Ontañón, ‘The combinatorial multi-armed bandit problem and its application to real-time strategy games’, in *AIIDE*, (2013).
- [17] A. Saffidine, H. Finnsson, and M. Buro, ‘Alpha-beta pruning for games with simultaneous moves’, 2012.
- [18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

# Relaxation Heuristics for Multiagent Planning

Michal Štolba<sup>1</sup> and Antonín Komenda<sup>2</sup>

stolba@agents.fel.cvut.cz, akomenda@tx.technion.ac.il

<sup>1</sup>Department of Computer Science and Engineering,  
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

<sup>2</sup>Faculty of Industrial Engineering and Management,  
Technion—Israel Institute of Technology, Haifa, Israel

## Abstract

Similarly to classical planning, in MA-STRIPS multiagent planning, heuristics significantly improve efficiency of search-based planners. Heuristics based on solving a relaxation of the original planning problem are intensively studied and well understood. In particular, frequently used is the delete relaxation, where all delete effects of actions are omitted. In this paper, we present a unified view on distribution of delete relaxation heuristics for multiagent planning.

Until recently, the most common approach to adaptation of heuristics for multiagent planning was to compute the heuristic estimate using only a projection of the problem for a single agent. In this paper, we place such approach in context with techniques which allow sharing more information among the agents and thus improve the heuristic estimates. We thoroughly experimentally evaluate properties of our distribution of *additive*, *max* and *Fast-Forward* relaxation heuristics in a planner based on distributed Best-First Search. The best performing distributed relaxation heuristics favorably compares to a state-of-the-art MA-STRIPS planner in terms of benchmark problem coverage. Finally, we analyze impact of limited agent interactions by means of recursion depth of the heuristic estimates.

## Introduction

Planning in a shared deterministic environment for and by a team of cooperative agents with common goals is a natural extension of classical planning. To model such planning problems, (Brafman and Domshlak 2008) proposed a multiagent extension of the classical STRIPS formalization, MA-STRIPS. The model presumes a set of cooperative agents defined by their capabilities in form of a set of actions partitioned from the original planning problem. In general, not all the agents need to (or even can not) consider the complete planning problem, therefore only subsets of facts and actions are marked as *public* and known to the whole team.

In recent years, several multiagent planning techniques solving MA-STRIPS problems were proposed. One focusing on optimality, scalability and efficiency

was proposed by (Nissim and Brafman 2012). It adopted one of the currently most successful approaches in classical planning—Best-First Search with highly informed automatically derived heuristics. The heuristics used were LM-cut (Helmert and Domshlak 2009) with pathmax equation and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007), both admissible as the approach aimed at optimal planning. The heuristics used only local information of the respective agents, effectively working only with their own facts and actions and public actions of other agents. (Crosby, Rovatsos, and Petrick 2013) proposed a centralized planning approach based on multiagent decomposition and local heuristic estimates. The approach used delete relaxation, particularly the Fast-Forward (FF) (Hoffmann and Nebel 2001) heuristics and focused on satisfiability. Another approach proposed by (Borrajó 2013) can in principle end up as planning with a global heuristic estimations, however requires private information of other agents<sup>1</sup>. On the contrary, the approach by (Torreño, Onaindia, and Sapena 2013) preserves private knowledge and proposes distributed heuristic estimate, however it is not based on relaxation of the original problem, but on Domain Transition Graphs (Helmert 2006). In discussion of (Nissim and Brafman 2012), the authors state that “*the greatest practical challenge [...] is that of computing a global heuristic by a distributed system*”. A recent work by (Štolba and Komenda 2013) targeted this challenge for distributed Fast-Forward heuristic with focus on obtaining the same estimates as by a centralized solution, rather than searching for a general solution for wide variety of relaxation heuristics.

In this work, we focus on distribution of the general principle of delete relaxation heuristics in MA-STRIPS planning with state-of-the-art implementation approaches. We evaluate properties of such distributed heuristics both from computational and communication perspectives and analyze the quality of the heuristics based on estimation depth in a sense of participating agents, i.e., *agent coupling relaxation*.

<sup>1</sup>The proposed solution used obfuscation of the private information, which can be prohibited in cases where even the “structure” of the information has not to be revealed.

## Multiagent Planning

A MA-STRIPS (Brafman and Domshlak 2008) planning problem is a quadruple  $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_g \rangle$ , where  $\mathcal{L}$  is a set of propositions,  $\mathcal{A}$  is a set of cooperative *agents*  $\alpha_1, \dots, \alpha_{|\mathcal{A}|}$ ,  $s_0$  is an initial state and  $S_g$  is a set of goal states. A *state*  $s \subseteq \mathcal{L}$  is a set of atoms from a finite set of propositions  $\mathcal{L} = \{p_1, \dots, p_m\}$  which hold in  $s$ . An *action* is a tuple  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$ , where  $a$  is a unique action label and  $\text{pre}(a), \text{add}(a), \text{del}(a)$  respectively denote the sets of preconditions, add effects and delete effects of  $a$  from  $\mathcal{L}$ .

An *agent*  $\alpha = \{a_1, \dots, a_n\}$  is characterized precisely by its capabilities, a finite repertoire of actions it can preform in the environment. MA-STRIPS problems distinguish between the *public* and *internal* (or *private*) facts and actions. Let  $\text{atoms}(a) = \text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)$  and similarly  $\text{atoms}(\alpha) = \bigcup_{a \in \alpha} \text{atoms}(a)$ . An  $\alpha$ -internal subset of all facts  $\mathcal{L}$  of agent  $\alpha$  will be denoted as  $\mathcal{L}^{\alpha-\text{int}}$ , where  $\mathcal{L}^{\alpha-\text{int}} = \text{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \text{atoms}(\beta)$  and a subset of all public facts as  $\mathcal{L}^{\text{pub}} = \mathcal{L} \setminus \bigcup_{\alpha \in \mathcal{A}} \mathcal{L}^{\alpha-\text{int}}$ . All facts relevant for one particular agent  $\alpha$  are denoted as  $\mathcal{L}^\alpha = \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$  and a *projection* of a state  $s^\alpha$  to an agent  $\alpha$  is a subset of a global state  $s$  containing only public facts and  $\alpha$ -internal facts, formally  $s^\alpha = s \cap \mathcal{L}^\alpha$ . The set of *public actions* of agent  $\alpha$  is defined as  $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \text{atoms}(a) \cap \mathcal{L}^{\text{pub}} \neq \emptyset\}$  and *internal actions* as  $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$ . The symbol  $a^\alpha$  will denote a projection of action  $a \in \beta, \beta \neq \alpha$  for agent  $\alpha$ , i.e., action stripped of all propositions of other agents, formally  $\text{atoms}(a^\alpha) = \text{atoms}(a) \cap \mathcal{L}^\alpha$ . Finally,  $\alpha^{\text{proj}}$  will denote the set of all public actions of other agents  $\mathcal{A} \setminus \alpha$  projected for agent  $\alpha$ .

Note that all actions of an agent  $\alpha$  uses only agent's facts, formally  $\forall a \in \alpha : \text{atoms}(a) \subseteq \mathcal{L}^\alpha$  by definition in (Brafman and Domshlak 2008). The goal set  $S_g$  of a multiagent planning problem will be treated as public (Nissim and Brafman 2012), therefore all goal-achieving actions are public.

### Multiagent Best-First Search

The planning algorithm we assume for the further analysis of the distributed heuristics is based on a multi-agent Best-First Search (MA-BFS) derived from the work in (Nissim and Brafman 2012) with deferred-evaluation of (lazy) heuristics. It is outlined in Algorithm 1. The MA-BFS algorithm is based on the standard textbook BFS. Firstly, the *Open* list is initialized with the initial state  $s_i$  and the *Closed* list is empty (line 1). In an infinite loop (lines 2–15), the *Open* list is polled to obtain state  $s$  (line 4). If  $s$  was not processed yet, it is marked as closed (lines 5, 6). If  $s$  is a goal state, the search is terminated and the plan is reconstructed (lines 7–9), otherwise, the heuristic estimate of the state is computed (line 10).

The MA-BFS differs in several aspects. Because the computation of an heuristic estimate may invoke communication among multiple agents, the heuristic estimators are asynchronous. The estimator is called with

---

### Algorithm 1 Multiagent Best-First Search

---

**Input:** Initial state  $s_i$ , goal  $S_g \subseteq \mathcal{L}$ , set of agent's actions  $\alpha$ , heuristic estimator  $\mathcal{H}$ .

```

1: Open  $\leftarrow \{s_i\}, \textit{Closed} \leftarrow \emptyset$ 
2: while true do
3:   if Open  $\neq \emptyset$  then
4:      $s \leftarrow \text{poll}(\textit{Open})$ 
5:     if  $s \notin \textit{Closed}$  then
6:        $\textit{Closed} \leftarrow \textit{Closed} \cup \{s\}$ 
7:       if  $s$  unifies with  $S_g$  then
8:         reconstruct the plan
9:       end if
10:       $\mathcal{H}(s, \text{heuristicComputedCallback}(s, h))$ 
11:     end if
12:   end if
13:   process heuristic messages
14:   process search messages
15: end while
```

---

```

16: heuristicComputedCallback( $s, h$ ):
17:   set the heuristic estimate of  $s$  to  $h$ 
18:   if  $h \neq \infty$  then
19:     if  $s$  obtained by  $a$  s.t.  $a \in \alpha^{\text{pub}}$  then
20:       send state  $s$ 
21:     end if
22:      $\textit{Open} \leftarrow \textit{Open} \cup \text{expand}(s, \alpha)$ 
23:   end if
```

---

a callback in its parameter and the main loop immediately continues (line 10). When the heuristic estimation is finished, the callback (lines 16–23) is invoked and it performs the standard procedure of setting the heuristic value (line 17) and expanding the state using applicable actions (line 22). In addition to that, if the state was obtained by expanding a public action, the state is sent to all other agents.

Finally at the end of each loop (lines 13 and 14) the messages related to the heuristic estimation (see next section) and the messages containing states sent by other agents (originating from line 20) are processed. The received states are added to the *Open* list. Note, that in MA-STRIPS a global state  $s \subseteq \mathcal{L}$  is seen by sending agent  $\alpha$  as a projection  $s^\alpha$  and by receiving agent  $\beta$  as a projection  $s^\beta$ .

### Distribution of Relaxation Heuristics

One general approach to compute heuristic estimates is to compute a solution of a *relaxed planning problem*. Such problems have some constraints removed (relaxed) in order to make it easier to solve them. One of well-known and thoroughly studied relaxations is obtained by removing the delete effects of all actions. Our motivation in this paper is to extend this concept to multiagent planning, therefore we will focus on classical delete relaxation heuristics: (i) inadmissible  $h_{add}$ , (ii) admissible  $h_{max}$  both introduced in (Bonet and Geffner 1999) and (iii) inadmissible  $h_{FF}$ , which was presented

in (Hoffmann and Nebel 2001). In the following subsections, we will present efficient multiagent distributions of those three heuristics.

### Distribution of $h_{add}$ and $h_{max}$

Both additive and max heuristics work on a very similar principle and are typically formalized as a set of recursive equations, such the following for  $h_{add}$ :

$$h_{add}(P, s) = \sum_{p \in P} h_{add}(p, s) \quad (1)$$

$$h_{add}(p, s) = \begin{cases} 0 & \text{if } p \in s \\ h_{add}(\text{argmin}_{a \in O(p)} [h_{add}(a, s)], s) & \text{otherwise} \end{cases} \quad (2)$$

$$h_{add}(a, s) = \text{cost}(a) + h_{add}(\text{pre}(a), s), \quad (3)$$

where  $P$  is a set of propositions (i.e., goal or action preconditions),  $s$  is a state,  $a$  is an action and  $O(p)$  is a set of actions which achieve  $p$ , formally  $O(p) = \{a \in \alpha \mid p \in \text{add}(a)\}$ . The equations for  $h_{max}$  are the same except for Equation 1 where is a *max* function instead of *sum*, therefore everything we state about  $h_{add}$  applies similarly to  $h_{max}$ .

In the multiagent setting, some of the actions in the *argmin* clause in Equation 2, where we are choosing the minimal cost action among actions achieving the proposition  $p$ , may be projections of other agent's public actions ( $a \in \alpha \cup \alpha^{proj}$ ). Let  $\alpha$  be the agent currently computing the heuristic estimate of the state  $s$  and  $\beta$  be some other agent. Let for some proposition  $p$  exist an action  $a \in \beta$  s.t.  $a^\alpha \in O(p)$ . In such case, there are two options how to handle the situation.

One option is to ignore the fact that the action is a projection and continue as if it was an ordinary action. This way, we may leave out some preconditions of the action (private to the owning agent), but we still get lower or equal estimate of the action cost (by including the private preconditions we can only increase the cost). We will denote the approach as a *projected heuristic*. Projected heuristics require no communication at all.

The other option is to always compute true estimate. In order to do so, the agent  $\alpha$  sends request  $r = \langle a^\alpha, s \rangle$  to agent  $\beta$  to obtain true estimate of the cost of the action  $a^\alpha$ . Upon receiving the request, agent  $\beta$  calls  $h_{add}(\text{pre}(a), s)$  and returns the result in a reply. It is obvious that in order to compute the heuristic estimate, agent  $\beta$  may need to send similar requests to other agents, or even back to agent  $\alpha$ . This way, we end up with a distributed recursive algorithm, which gives exactly the same results as a centralized  $h_{add}$  on a global problem  $\Pi_G = \langle \mathcal{L}, \bigcup_{\alpha \in \mathcal{A}} \alpha, s_0, S_g \rangle$ , since for every projection  $a^\alpha$  of action  $a \in \beta$ , the true cost of  $a$  is obtained from the agent  $\beta$ .

A middle ground between the presented two extremes is to limit the recursion depth  $\delta$ . If the *maximum recursion depth*  $\delta_{max}$  is reached, all projected actions are evaluated without sending any further requests. This effectively means introduction of another relax-

---

### Algorithm 2 Distributed Relaxed Exploration

---

**Input:** Boolean flag  $r$  (true when first called), global exploration queue  $\mathcal{Q}$

**relaxedExploration( $r$ ):**

```

1: while  $\mathcal{Q} \neq \emptyset$  do
2:    $p \leftarrow \text{poll}(\mathcal{Q})$ 
3:   if  $p \in \text{goal}$  and  $\text{achieved}(p') : \forall p' \in \text{goal}$  then
4:     return
5:   end if
6:    $O_p \leftarrow \{a \in \alpha \cup \alpha^{proj} \mid p \in \text{pre}(a)\}$ 
7:   for all  $a \in O_p$  do
8:     increment  $\text{cost}(p)$  by  $\text{cost}(a)$ 
9:     if  $\text{achieved}(p') : \forall p' \in \text{pre}(a)$  then
10:      enqueueProposition( $a, \text{eff}(a), r$ )
11:    end if
12:   end for
13: end while

```

---

**Input:** Action  $a$ , proposition  $p$ , Boolean flag  $r$

**enqueueProposition( $a, p, r$ ):**

```

14: if  $\text{cost}(p) = \perp$  or  $\text{cost}(p) > \text{cost}(a)$  then
15:    $\text{cost}(p) \leftarrow \text{cost}(a)$ 
16:   if  $r$  and  $a \in \alpha^{proj}$  then
17:     send request message  $M_{req} = \langle s, a, 0 \rangle$ 
       to  $\text{owner}(a)$ ,
       process the reply by
       receiveReplyEnqueueCallback( $p, -$ )
18:   else
19:      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{p\}$ 
20:   end if
21: end if

```

---

**Input:** Heuristic estimate  $h$ , proposition  $p$  (set from enqueueProposition)

**receiveReplyEnqueueCallback( $p, h$ ):**

```

22: if  $\text{cost}(p) > h$  then
23:    $\text{cost}(p) \leftarrow h$ 
24:    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{p\}$ 
25: end if
26: relaxedExploration(false)
27: if no unresolved requests then
28:   return compute the total cost
29: end if

```

---

ation of the original problem where the interaction between agents is limited—the *agent coupling relaxation*. Such heuristic estimation is always lower or equal than would be the heuristic estimation in the global problem  $\Pi_G$  using a centralized heuristic estimator, because ignoring preconditions of an action in its projection can never increase the cost of the action. By limiting the recursion depth to  $\delta_{max} = 0$ , we return back to the *projected heuristic*, where all interactions between agents are relaxed away.

**Relaxed exploration.** Although the definition of  $h_{add}$  by a set of recursive equations is intuitively clear and provides good theoretical background, in practice, the recursive functions are not typically used. Recursive calls have their limitations in the call stack and converting such recursion, where the recursive call is within a complex function *argmin*, into iteration is possible, but rather cumbersome. Instead, the idea of *relaxed exploration* is typically utilized.

The *relaxed exploration* is in fact a reachability analysis of the relaxed planning problem, which can be conveniently seen as building a *relaxed planing graph* (RPG). A relaxed planning graph is a layered (alternating fact and action layers) directed graph. In its first layer it contains all facts which hold in the initial state, the next layer contains all actions of which preconditions are satisfied in the previous layer (and noop actions), the next layer contains all (add) effects of the actions from previous layer and so forth. In practice, a RPG is not built explicitly, but the exploration is achieved via effective representation we will refer to as an *exploration queue* (based on the Fast-Downward planning system (Helmert 2006)).

The *exploration queue* considers only *unary actions*—actions which have a single proposition as an add effect (any relaxed problem can be converted so it contains only unary actions). The *exploration queue* is supported by a data structure representing the precondition-of and achieved-by relations. The queue is initialized with the propositions which are true in the state  $s$ . Until the queue is empty, a proposition  $p$  is polled, it is checked whether  $p$  is a goal proposition and if so, whether all goals are satisfied. If not, for each action that depends on  $p$  ( $p \in \text{pre}(a)$  where  $a \in \alpha \cup \alpha^{proj}$ ), the action cost is incremented by the cost of proposition  $p$  (that is either added for  $h_{add}$ , or maxed for  $h_{max}$ ) and if there are no more unsatisfied preconditions of the action, the action is applied. The process is detailed in Algorithm 2, lines 1–13. Thanks to the sole use of *unary operators*, the application of an action  $a$  can be interpreted as adding the (only one) proposition  $p = \text{eff}(a)$  to the *exploration queue*, thus the procedure is named *enqueueProposition* (line 10).

The effectiveness of this approach lays in fact, that during the relaxed exploration, cost estimates of facts and actions can be conveniently computed and once all goal facts are reached, the heuristic can be computed by simple *sum* or *max* of costs of all goal facts respectively.

**Distributed relaxed exploration.** An algorithm capable of building RPGs in a distributed manner was presented in (Štolba and Komenda 2013). The major drawback of the used approach was the necessity to build the RPG for each state by all agents at once, thus preventing the search to run independently in parallel. It was shown, that the resulting heuristic estimate is equal to the centralized estimate. In this paper, we will not place the requirement of obtaining the same value as in the centralized variant, which will allow us

---

#### Algorithm 3 Request Processing

---

**Input:** Request message  $M_{req} = \langle s, a, \delta \rangle$ , where  $s$  is state,  $a$  action,  $\delta$  recursion depth,  $\beta$  the sender

**processRequest**( $M_{req} = \langle s, a, \delta \rangle, \beta$ ):

- 1:  $\mathcal{Q} \leftarrow \{s\}$
  - 2: relaxedExploration(false)
  - 3:  $h \leftarrow$  compute the total cost
  - 4:  $P \leftarrow$  mark public actions
  - 5: send reply message  $M_{re} = \langle h, P, \delta \rangle$  to  $\beta$
- 

---

#### Algorithm 4 Reply Processing

---

**Input:** Reply message  $M_{re} = \langle h, P, \delta \rangle$ , where  $h$  is heuristic estimate,  $P$  set of actions,  $\delta$  recursion depth

**processReply**( $M_{re} = \langle h, P, \delta \rangle$ ):

- 1: **if**  $\delta < \delta_{max}$  **then**
  - 2:  $h_{sum} \leftarrow h$
  - 3: **for all**  $a \in P$  **do**
  - 4: send request message  $M_{req} = \langle s, a, \delta + 1 \rangle$   
to *owner*( $a$ ),  
process the reply by  
receiveReplyCallback( $h$ )
  - 5: **end for**
  - 6: **end if**
  - 7: receiveReplyEnqueueCallback( $_, h$ )
- 

**Input:** Heuristic estimate  $h$

**receiveReplyCallback**( $h$ ):

- 8:  $h_{sum} \leftarrow h_{sum} + h$
  - 9: **if** all replies received **then**
  - 10: receiveReplyEnqueueCallback( $_, h_{sum}$ )
  - 11: **end if**
- 

to build a much more efficient algorithm. The algorithm is based on building the *exploration queue* and requesting other agents when projections of their actions are encountered. Moreover the presented algorithm allows for precise control of the recursion depth and thus enables us to trade-off the estimation precision with the computation and communication complexity.

The basic process of building the *exploration queue*  $\mathcal{Q}$  is similar to the centralized version as described in the previous section. The main principle of the distributed process is that whenever a projection of some other agent's action should be applied (and its effect added to the queue), a request is sent to the owner of the action to obtain its true cost. The effect of the action is added to the queue only after the reply is received. Note, that when computing the reply, the agent may need to send requests as well, thus ending up with a distributed recursion. In order to effectively handle the recursion it is flattened so that all requests are sent by the initiator agent and the replies are augmented

with the parameters of the next recursive call.

The exploration part of the algorithm is shown in Algorithm 2, whereas Algorithms 3 and 4 details the inter-agent communication. The entry point of the algorithm is the **relaxedExploration** procedure. First, it is invoked with the  $r$  parameter set to **true**, indicating, that whenever a projected action is encountered, a request is sent to its owner.

The main difference between the centralized and distributed approaches lays in the **enqueueProposition** procedure. If the cost of the action improves the current cost of the proposition, the cost of the proposition is set equal to the cost of the action, as usual, but if the action  $a \in \alpha^{proj}$  is a projection and sending of requests is enabled, i.e.  $r = \text{true}$ , a request message  $M_{req} = \langle s, a, \delta \rangle$ , where  $s$  is the current state,  $a$  is the action and initial recursion depth  $\delta = 0$ , is sent to the owner of the action  $a$ , i.e. agent  $\beta$ . Otherwise, the proposition is added to the *exploration queue*.

**Processing the messages.** When the request message is received by the agent  $\beta$  (see Algorithm 3, **processRequest**), the *relaxed exploration* is run with the goal being the preconditions of the requested action  $a$  and without sending any requests, i.e.,  $r = \text{false}$ . After finishing the exploration, public actions which have contributed to the resulting heuristic estimate are determined (line 4). In principle, the procedure is similar to extracting a relaxed plan in the FF heuristic. A reply  $M_{re} = \langle h, P, \delta \rangle$  is sent, where  $h$  is the computed heuristic value,  $P$  is the set of the contributing public actions and  $\delta$  is the current recursion depth.

Receiving the reply from agent  $\beta$  is managed by procedure **processReply** in Algorithm 4. If the recursion depth has already reached the limit  $\delta > \delta_{max}$ , the original `receiveReplyEnqueueCallback(p, h)` from Algorithm 3 for action  $a$  is called, the cost estimate of proposition  $p$  is finalized and  $p$  is added to the *exploration queue*. Since the messaging process is asynchronous, the original *relaxed exploration* has already terminated, therefore it is started again (line 26), with the original data structures and with the newly evaluated proposition added to the queue. When the exploration is finished and there are no pending requests, the final heuristic estimate is computed depending on the actual heuristic (*sum* or *max*) and is returned via a callback to the search, so that the evaluated state can be expanded.

Otherwise, if  $\delta \leq \delta_{max}$ , agent  $\alpha$  iterates through all actions  $a' \in P$  and sends requests to their respective owners. The heuristic estimate received in each reply is added to the shared  $h_{sum}$ . When all replies are received (the replies undergo the same procedure, if there are any other public actions involved) and all costs are added together, again the `receiveReplyEnqueueCallback(p, h)` from Algorithm 2 is called with  $h = h_{sum}$ .

The **processReply** procedure stands for the distributed recursion, but the deeper recursive call is not called by the agent  $\beta$ , but the parameters of the recursion (the set of actions  $P$  which should be resolved next)

are sent back to the initiator agent. This is rather an optimization which prevent us from the need of having multiple heuristic evaluation contexts needed to handle multiple interwoven request/reply traces. Each context would need to have separate instance of the *exploration queue* data structure, which would present major inefficiency. Instead, the initiator agent is responsible for tracking the recursion and the replying agent only processes one reply at a time, locally, without sending any requests. Therefore, each agent needs to have only two instances of the *exploration queue*, one used to compute their own heuristic estimates (and possibly send requests and await replies), and one used to compute the local estimates for the replies.

### Distribution of $h_{FF}$

The Fast-Forward  $h_{FF}$  heuristic is not directly based on estimation of the cost of actions in the relaxed problem, but on actually finding a plan solving the relaxed problem (a relaxed plan or RP). The heuristic is not typically described using recursive equations, but the implementation based on *relaxed exploration* can be easily reused. The difference is, that the evaluation does not end when the exploration is finished (all goal propositions have been reached), but continues with the relaxed plan extraction. The extraction of RP starts with the goal propositions and traverses the data structure towards the initial state, while marking the relaxed plan.

Since the algorithm is implementation-wise very similar to the  $h_{add}$  heuristic, one of the possible approaches to distribution of  $h_{FF}$  is to perform the distributed *relaxed exploration* exactly as in  $h_{add}$  and simply add RP extraction routine at the end of the heuristic evaluation (as part of the total cost computation). Another approach was introduced in (Štolba and Komenda 2013) as lazy multiagent FF heuristic  $h_{lazyFF}$ , which we have adopted and compared with the previously described approach and both additive and max heuristics.

Our version of the lazy FF algorithm starts with local building of the *exploration queue*. When all goal propositions are reached, a relaxed plan  $\pi'$  is extracted. For all actions  $a \in \pi'$ , which are projections  $a \in \alpha^{proj}$ , request message  $M_{req} = \langle s, a, \delta \rangle$  is sent to the owning agent  $\beta = \text{owner}(a)$  of the action  $a$ . When agent  $\beta$  receives the request, he constructs a local relaxed plan from state  $s$  (by local *relaxed exploration* and local RP extraction without sending any requests), satisfying the preconditions (both public and private) of the action  $a$ . Then, agent  $\beta$  sends a reply  $M_{re} = \langle h, P, \delta \rangle$ , where  $h$  is the length of the relaxed plan and  $P$  is a set of projected actions contained in the plan. When the reply is received by agent  $\alpha$ , the algorithm iterates through all actions  $a' \in P$  and sends requests to their respective owners. Each of the requests undergo the same procedure as the original request, adding the returned heuristic estimates to the resulting  $h_{sum}$ . When all requests are processed,  $h_{sum}$  is added to the length of the local relaxed plan of agent  $\alpha$  and returned via callback as the heuristic estimate of state  $s$ .

prob. ( $ \mathcal{A} $ )	$\delta_{max}$	$h_{FF}$			$h_{add}$			$h_{max}$			$h_{lazyFF}$		
		$t[s]$	$e[k-]$	$b[MB]$	$t[s]$	$e[k-]$	$b[MB]$	$t[s]$	$e[k-]$	$b[MB]$	$t[s]$	$e[k-]$	$b[MB]$
Rov8 (4)	1	1.2	0.4	0.5	1.2	0.5	0.7	<b>1.1</b>	0.4	0.5	—	—	—
Rov8 (4)	$\infty$	<b>1.1</b>	0.5	0.7	1.2	0.6	0.8	<b>1.1</b>	0.5	0.6	—	—	—
Rov12 (4)	0	70.9	4617.3	35.7	40.7	3939.5	31.2	—	—	—	57	4583.5	35.6
Rov12 (4)	1	1.2	0.8	0.7	1.3	0.3	0.2	<b>1.1</b>	0.3	0.3	—	—	—
Rov12 (4)	$\infty$	<b>1.1</b>	0.4	0.4	1.2	0.7	0.6	1.2	0.3	0.3	—	—	—
Rov14 (4)	1	21	112.1	157.4	18.7	101.9	136.6	21.9	127.2	170.6	—	—	—
Rov14 (4)	$\infty$	23	124.7	175.1	19.2	98.5	138.7	<b>16.1</b>	90.6	127.6	—	—	—
Sat9 (5)	1	3.2	6.2	14.7	3.2	7.8	17.7	3.1	6.9	15.6	3.1	9.8	10.3
Sat9 (5)	$\infty$	3.7	10.6	25	3.5	7.4	17.4	3.3	6.2	14.6	<b>3</b>	7.4	8.4
Sat10 (5)	1	3.7	4.1	9.6	<b>3.4</b>	3.2	7.1	3.4	2.4	5.3	4.1	13.5	6.6
Sat10 (5)	$\infty$	3.7	4.3	9.9	4	8.2	19	3.3	2.3	5.3	3.9	7.8	6
Sat* (14)	1	69.2	9.3	36.9	68	8.7	33.4	69	9	34.8	<b>60.6</b>	9	17.5
Sat* (14)	$\infty$	69	9.3	36.7	68.5	9.3	37	68.5	9	35.7	61.3	9.7	18.4
Sat* (16)	1	133.8	11.1	57.5	136.6	11.7	59.4	133.6	10.7	54.1	126.4	12.8	31
Sat* (16)	$\infty$	132.7	11.1	57.8	137.3	12.4	64.6	133.5	11.1	57.8	<b>124.5</b>	12.3	28.8
Log* (6)	0	<b>0.7</b>	6.8	0	0.7	5.7	0	0.7	7	0	<b>0.7</b>	7.2	0
Log* (6)	1	1.2	0.5	0.3	1.6	1.3	0.6	1.2	0.7	0.3	1.6	5.4	0.6
Log* (6)	$\infty$	1.1	0.4	0.2	1.3	0.8	0.5	1.2	0.5	0.3	1.4	0.6	0.8
CP* (6)	0	1.8	136.1	2.2	1.6	99.5	1.7	2.6	351.5	5.5	1.8	137.7	2.2
CP* (6)	1	1.7	127.5	2.1	7.3	72.1	51.6	10	100.2	71.8	5.6	43.8	66.1
CP* (6)	$\infty$	1.7	122.5	2	<b>1.4</b>	76.2	1.3	2.6	352.6	5.4	44.3	91.8	973.5
CP* (7)	0	2.2	183.2	2.4	2.2	223.1	3.2	6.3	1252.5	15.7	2.3	205	2.7
CP* (7)	1	<b>1.9</b>	162.1	2.2	18.3	248.1	150.8	35.5	451.4	274.5	50.4	371.2	738.6
CP* (7)	$\infty$	2	188.9	2.5	2.1	225.2	3.2	6.3	1255.6	15.5	160.9	249.5	249.5
Sok* (2)	0	1.6	8.5	0.5	1.5	7.7	0.5	1.7	11.7	0.7	1.6	8.8	0.6
Sok* (2)	1	1.5	7.6	0.5	17.1	22	66.8	3.9	4.3	12.1	4.3	12.9	24.1
Sok* (2)	$\infty$	1.5	8.3	0.5	<b>1.4</b>	7.8	0.5	1.6	11.7	0.7	—	—	—

Table 1: Comparison of all presented heuristics for selected problems and metrics. The planning time metrics  $t$  is in seconds, the explored states  $e$  in thousands of states and the communicated information  $b$  in megabytes. Abbreviations: Rov = rovers, Sat = satellites, Log = logistics, CP = cooperative path-finding, Sok = sokoban.

The recursion depth of the heuristic estimate can be limited in a similar manner as in the add/max heuristics. Whenever a request should be sent and the maximum recursion limit  $\delta_{max}$  has been reached, the request is not sent and the possible relaxed sub-plan is ignored.

## Experiments

To analyze properties of the proposed distributed heuristics and their implementations, we have prepared a set of experiments covering various efficiency aspects of the heuristics.

All experiments were performed on FX-8150 8-core processor at 3.6GHz, each run limited to 8GB of RAM and 10 minutes. Each measurement is a mean from 5 runs. We have used the translation to SAS+ formalism and preprocessing from the Fast-Downward planning system (Helmert 2006) and new implementation of the search and heuristic estimators.

### Initial comparison

The first batch of experiments focused on two classical planning metrics used in comparison of heuristic efficiency: planning time  $t$  and number of explored states

$e$ . Those metrics were supplied by a multiagent metric of communicated bytes  $b$  among the agents during the planning process. Used planning problems stem from IPC domains modified for multiagent planning as presented, e.g., in (Nissim and Brafman 2012). The problems with \* in their names were either based on IPC domains, but simplified, or other state-of-the-art multiagent benchmarks, e.g., from (Komenda, Novák, and Pěchouček 2013). The recursion depth was limited to three values  $\delta_{max} = \{0, 1, \infty\}$  as other settings of  $\delta_{max}$  showed similar results. Missing rows were not successfully planned with any of the tested heuristics.

The results are summarized in Table 1. No single heuristic and  $\delta_{max}$  dominates the other ones. In Rovers, the most successful seems to be  $h_{max}$ . In Satellites, good performing is  $h_{lazyFF}$ , but it loses in other domains, where it does not solve some problems at all. The Logistics is dominated by  $h_{FF}$  and in Cooperative Path-Finding and Sokoban, the best are  $h_{FF}$  and  $h_{add}$ .

### Problem coverage

In this experiment, we have evaluated the coverage of all the described heuristics ( $h_{add}, h_{max}, h_{FF}$  and  $h_{lazyFF}$ ) with the maximum recursion depth  $\delta_{max}$  set to 0,

$\delta_{max}$	0	1	2	4	$\infty$
$h_{FF}$	35 / 7	38 / 15.4	38 / 15	38 / 14.4	38 / 15
$h_{add}$	35 / 7	38 / 14	38 / 14	38 / 14	38 / 14
$h_{max}$	35 / 3.2	38 / 14	38 / 14	38 / 14	38 / 14
$h_{lazyFF}$	35.2 / 6.8	38 / 8	36.2 / 8	36.5 / 8	36.8 / 8

Table 2: Coverage for various heuristics and recursions depth  $\delta_{max}$ . The results are in the form of multiagent domains / IPC domains.

1, 2, 4 and  $\infty$ . The coverage has been evaluated over two sets of benchmarks. First set consists of 40 specifically multiagent problems, which are typically not that combinatorially hard, but contain more agents (taken from (Štolba and Komenda 2013) and (Komenda, Novák, and Pěchouček 2013)). Second set consists of 21 problems converted directly from IPC benchmarks (as in (Nissim and Brafman 2012)), which are typically much combinatorially harder, but with less agents. The results are summarized in Table 2.

The results show clear dominance of  $h_{FF}$ , but interestingly the other distribution approach of the Fast-Forward heuristic,  $h_{lazyFF}$ , is on the other side of the spectrum. This is most probably because one of the biggest strengths of the FF heuristic, compared to other delete relaxation heuristics used here, is that it does not suffer from *over-counting* (one action is included in the estimate several times) thanks to the explicit relaxed plan extraction. In the  $h_{lazyFF}$ , we partially lose this advantage, because when sending reply, only the length of the plan is sent. Therefore, single action can be included several times in multiple replies from single agent, or even multiple agents.

Another message the table conveys is that the setting of  $\delta_{max} = 0$  is dominated by other values. This may be due to the choice of the domains, the effect of various  $\delta_{max}$  settings is thoroughly analyzed in the next set of experiments. Also, various settings of  $\delta_{max}$  for  $\delta_{max} > 0$  affect the coverage only marginally.

We can also state, that in the terms of coverage, the use of distributed heuristics compares favorably to the state of the art, as in (Nissim and Brafman 2012), the planner using *projected heuristics* LM-cut and Merge&Shrink has a coverage of 11 resp. 12 problems, which was exceeded by all distributed heuristic with  $\delta_{max} > 0$ , except for  $h_{lazyFF}$ . But it is important to emphasize, that the comparison is not completely fair, as the approach used in (Nissim and Brafman 2012) is optimal and although we use the admissible  $h_{max}$  heuristic, our approach is not optimal. To do so requires special handling of the search termination, as detailed in (Nissim and Brafman 2012).

### Effect of the recursion depth

In the following set of experiments, we have evaluated the effect of changing the *maximal recursion depth*  $\delta_{max}$  on the speed and communication requirements of the

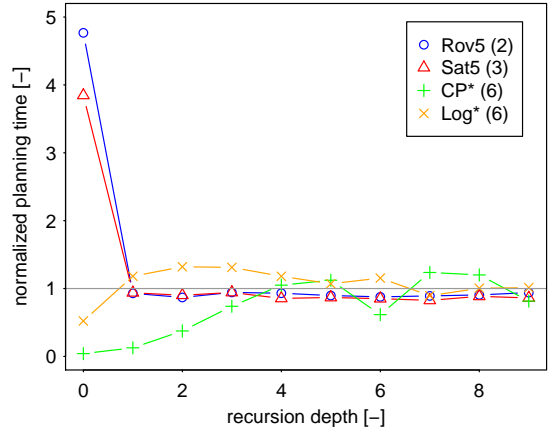


Figure 1: Planning time normalized to result for of  $\delta_{max} = \infty$  for  $h_{lazyFF}$  heuristics.

planning process. The data set was measured on four selected domains with varied couplings (rovers, satellites, cooperative path-finding and logistics), each represented by a single problem. The *maximal recursion depth* ranged from 0 (a *projected heuristic*) to 9, for comparison, the results were normalized against the result of run with  $\delta_{max} = \infty$ .

By coupling, we understand the concept formalized in (Brafman and Domshlak 2008), which can be rephrased as “the more interactions must take place among the agents in order to solve the problem, the more coupled the problem is”—at one extreme there are problems, where all actions interact with other agents (containing only public actions) meaning *full coupling*. In problems of the other extreme, the agents can solve their individual problems without any interaction. Because of our decision to treat all goals as public, we cannot achieve *full decoupling*—at least goal-achieving actions are public and thus causing some level of coupling. The experimental domains were chosen such that rovers and satellites are loosely coupled. In satellites, only the assumption that all goal-achieving actions are public introduces some coupling, in rovers, there are also interacting preconditions among the goal-achieving actions. Logistics is quite balanced (private movement of agents and public handling of packages) and cooperative path-finding is fully coupled.

The experimental results for the  $h_{lazyFF}$  heuristic are plotted in Figures 1 and 2. In the fully coupled cooperative path-finding, the results are best for  $\delta_{max} = 0$  and are converging to the results for  $\delta_{max} = \infty$  as  $\delta_{max}$  grows. This is because in a fully coupled problem, all actions are public and in cooperative path-finding all their preconditions and effects are also public (which does not have to be always the case). Therefore each agent has complete information about the problem in form of the action projections ( $a^\alpha = a$  for all actions and agents) and the *projected heuristic* gives perfect es-



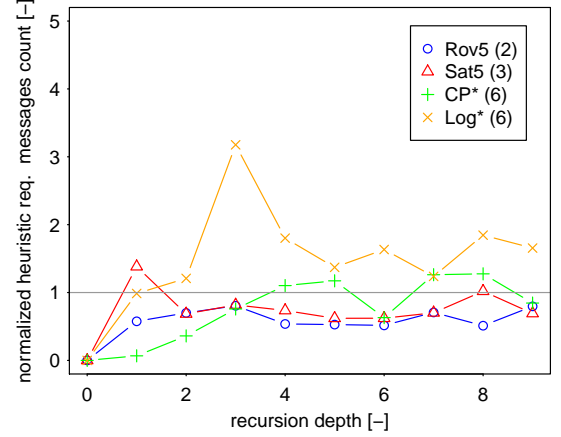
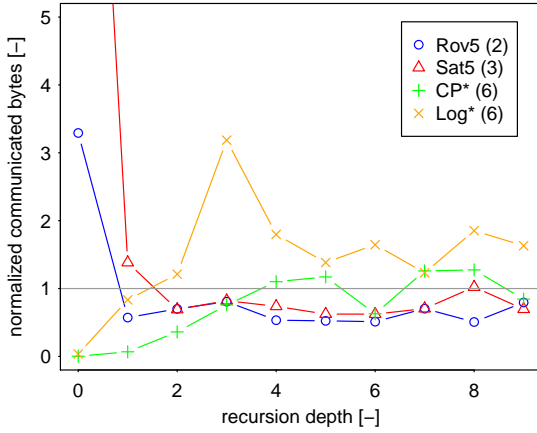


Figure 2: Communicated bytes and heuristic message requests normalized to  $\delta_{max} = \infty$  for  $h_{lazyFF}$  heuristics.

timate (the same as would global heuristic give). For  $\delta_{max} > 0$ , requests are sent for every projected action, causing more communication and computation without bringing any improvement to the heuristic estimate.

Completely different picture give the results for the loosely coupled problems. The results are significantly worse for  $\delta_{max} = 0$ , from  $\delta_{max} = 1$  they are practically equal to  $\delta_{max} = \infty$ . The solution of those problems typically consist of long private parts finished by a single public action (the goal achieving action). When estimated by a *projected heuristic*, the private parts of other agents get ignored and the estimates are thus much less informative. Even the fact, that when a state is expanded by a public action, it is sent with the original agent’s heuristic estimate, does not help, because estimation of states expanded further from such state ignore the information again. But even  $\delta_{max} = 1$  is enough to resolve this issue.

Lastly, in the balanced logistics problem, the  $\delta_{max} = 0$  estimates are rather good (but not as good as in the cooperative path-finding) and with growing  $\delta_{max}$ , the results converge towards  $\delta_{max} = \infty$ , but for  $0 < \delta_{max} < \infty$  the results are slightly worse. This may suggest, that as the coupling is balanced, it is best either to fully exploit the coupled part of the problem and use *projected heuristics*, or to rely on the decoupled part of the problem and employ the full recursion approach, depending on the exact balance.

The results for communication are in Figure 2. The left chart compares the total bytes communicated and shows the same tendencies as the planning time in Figure 1. In fact, limiting the interactions may lead to increased communication. The right table shows the data for heuristic requests, there we see the expected result for  $\delta_{max} = 0$ , where no requests are sent, otherwise the tendencies are surprisingly similar. This indicates, that the communication complexity is dominated by the search communication complexity (the longer the search takes, the more messages are passed).

Those rather unintuitive results suggest, that for tightly coupled problems, sharing of the information is not only less important, because the agents have most of the information in their problem projections, but may even lower the effectiveness because of redundant computation. On the other hand, for loosely coupled problems, the communication is vital, even if the communication is very limited. For balanced problems, both extremes are equally good. In general, it is hard to determine, which approach will yield the best results, but it is sensible to choose from either no communication  $\delta_{max} = 0$ , full communication  $\delta_{max} = \infty$ , or even communication limited to very low recursion depth limits, i.e.,  $\delta_{max} = 1$ . If we can expect some properties of the problems at hand, we can suggest preferred approach much easier—if we are *not* expecting loosely coupled problems,  $\delta_{max} = 0$  is the best choice, for *no* tightly coupled problems  $\delta_{max} = \infty$  and for *no* balanced problems,  $\delta_{max} = 1$  seems to be the best choice.

The results in the presented figures are for  $h_{lazyFF}$  mainly because they are the most illustrative, other heuristics follow the same patterns as described here.

## Final Remarks

We have proposed an efficient distribution approach for three classical delete-relaxation heuristics  $h_{add}$ ,  $h_{max}$  and  $h_{FF}$ . The heuristics were experimentally compared in a planner utilizing a multiagent Best-First Search on various multiagent planning problems stemming from classical IPC domains. The comparison comprised metrics of planning time and communication and expanded states, coverage comparison and comparison of the effect of changing maximum recursion depth.

The results did not show any single heuristic to be dominating others, but brought to light interesting and rather unintuitive conclusion. For tightly coupled problems, it is more efficient to limit the information sharing, whereas loosely coupled problems strongly benefit from the distributed heuristic estimation.

**Acknowledgments** This research was supported by the Czech Science Foundation (13-22125S), A. Komenda was supported in part by *USAF EOARD* (FA8655-12-1-2096), in part by a Technion fellowship.

## References

- Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *Lecture Notes in Computer Science*, 360–372. Springer.
- Borrajo, D. 2013. Plan sharing for multi-agent planning. In *Proc. of DMAP Workshop of ICAPS’13*, 57–65.
- Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS’08*, 28–35.
- Crosby, M.; Rovatsos, M.; and Petrick, R. 2013. Automated agent decomposition for classical planning.
- Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proceedings of ICAPS’09*.
- Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of ICAPS’07*, 176–183.
- Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. 14:253–302.
- Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*.
- Nissim, R., and Brafman, R. I. 2012. Multi-agent A\* for parallel and distributed systems. In *Proceedings of AAMAS’12*, 1265–1266.
- Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proc. of DMAP Workshop of ICAPS’13*, 75–83.
- Torreño, A.; Onaindia, E.; and Sapena, O. 2013. Fmap: a heuristic approach to cooperative multi-agent planning. In *Proc. of DMAP Workshop of ICAPS’13*, 84–92.

# Fast-Forward Heuristic for Multiagent Planning

Michal Štolba and Antonín Komenda

{stolba|komenda}@agents.fel.cvut.cz

Department of Computer Science and Engineering,  
Faculty of Electrical Engineering, Czech Technical University in Prague

## Abstract

Use of heuristics in search-based domain-independent deterministic multiagent planning is as important as in classical planning. In this work we propose a formal and an algorithmic adaptation of a well-known heuristic Fast-Forward into multiagent planning. Such treatment is important as it solves challenges in decentralization of this and other heuristics based on relaxation of the original planning problem. Such decentralization enables global heuristic estimates to be computed without exposing local information. Additionally, since Fast-Forward heuristic is based on relaxed planning, we propose a multiagent approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate different distribution strategies of the heuristic estimate.

## Introduction

In recent years the landscape of multiagent planning research has changed by Brafman and Domshlak’s formal treatment and promising complexity results of domain-independent deterministic multiagent planning (DMAP) (Brafman and Domshlak 2008) represented as an extension of STRIPS for more agents. An important piece of the puzzle was a decomposition of a planning problem common for all the agents. In principle, the ideas behind relate to the research of planning problem factorization and utilization of such for more efficient solving of classical planning problems. Therefore even for cooperative agents, it is reasonable to hide parts of the information used during planning from other agents as this helps (in loosely coupled problems) the agents to focus only on their parts of the problem.

After this publication, the community started to design and implement first planners using the principles of DMAP described in the Brafman and Domshlak’s paper. The first one from Nissim et al. (Nissim, Brafman, and Domshlak 2010) was built on distributed constraint satisfaction problem solver and a forward chaining planner. This approach precisely followed the ideas

in (Brafman and Domshlak 2008), however exposed a couple of issues making the approach incomparable in efficiency with current state-of-the-art implementations of classical planners. One of the issues was bad scalability with growing length of the coordination part of the resulting plans. Improvement of scalability was proposed in (Nissim and Brafman 2012) by leaving the DisCSP+Planning approach and moving to a principle which is currently the most successful in classical planning—A\* or variations on Best First Search (BFS) with highly informed automatically derived heuristics.

Since the motivation of (Nissim and Brafman 2012) was to propose an optimal planner (MA-A\*), the heuristics used were LM-cut (Helmert and Domshlak 2009) with pathmax equation and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007). In the distributed search approach, the heuristics were used only with local information of the respective agent, i.e., with its internal actions, its public actions and projections of other agents’ public actions. In discussion of (Nissim and Brafman 2012), the authors state that *“the greatest practical challenge [...] is that of computing a global heuristic by a distributed system”*, which is precisely our focus in this work. According to our knowledge, there is no work proposing efficient planners for DMAP not focused on optimality of the resulting plans. In the field of classical planning, on the other hand, the best performing planners as Fast Downward and LAMA incorporate a fast, but suboptimal search algorithm using non-admissible heuristics.

In this work we propose a formal and algorithmic adaptation of a well-known relaxation heuristic Fast-Forward  $h^{\text{FF}}$  (Hoffmann and Nebel 2001) into multiagent planning. We argue that such treatment is important as it demonstrates algorithmic challenges in decentralization of computation of  $h^{\text{FF}}$  and other related heuristics. Additionally, since the  $h^{\text{FF}}$  heuristic is based on relaxed planning, we propose a multiagent (MA) approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate two distribution strategies of the heuristic estimate against the local estimate.

## Multiagent Planning

We consider a number of *cooperative* and *coordinated* agents featuring distinct sets of capabilities (actions), which concurrently plan and execute their local plans in order to achieve a joint goal. The world wherein the agents act is *classical* and the actions are *deterministic*. The following formal preliminaries compactly restate the MA-STRIPS problem (Brafman and Domshlak 2008) required for the following sections.

A MA-STRIPS planning problem is a quadruple  $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_g \rangle$ , where  $\mathcal{L}$  is a set of propositions,  $\mathcal{A}$  is a set of *agents*  $\alpha_1, \dots, \alpha_{|\mathcal{A}|}$ ,  $s_0$  is an initial state and  $S_g$  is a set of goal states. A *state*  $s \subseteq \mathcal{L}$  is a set of atoms from a finite set of propositions  $\mathcal{L} = \{p_1, \dots, p_m\}$  which holds in  $s$ . An *action* an agent can perform is a tuple  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$ , where  $a$  is a unique action label and  $\text{pre}(a), \text{add}(a), \text{del}(a)$  respectively denote the sets of preconditions, add effects and delete effects of  $a$ , taken from  $\mathcal{L}$ . *Act* denotes the set of all actions in the multiagent planning problem  $\Pi$ , i.e.,  $\text{Act} = \bigcup_{\alpha \in \mathcal{A}} \alpha$ .

An *agent*  $\alpha = \{a_1, \dots, a_n\}$  is characterized precisely by its capabilities, a finite repertoire of actions  $a_i \in \text{Act}$  it can preform in the environment. MA-STRIPS problems distinguish between the *public* and *internal* facts and actions. Let  $\text{atoms}(a) = \text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)$  and similarly  $\text{atoms}(\alpha) = \bigcup_{a \in \alpha} \text{atoms}(a)$ . An  $\alpha$ -internal and public subset of all facts  $\mathcal{L}$  will be denoted as  $\mathcal{L}^{\alpha\text{-int}}$  and  $\mathcal{L}^{\text{pub}}$  respectively, where  $\mathcal{L}^{\alpha\text{-int}} = \text{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \text{atoms}(\beta)$  and  $\mathcal{L}^{\text{pub}} = \text{atoms}(\alpha) \setminus \mathcal{L}^{\alpha\text{-int}}$ . Facts relevant only for one agent  $\alpha$  are denoted as  $\mathcal{L}^\alpha = \mathcal{L}^{\alpha\text{-int}} \cup \mathcal{L}^{\text{pub}}$  and a *projection* of a state  $s^\alpha$  to an agent  $\alpha$  is a subset of a global state  $s$  containing only public facts and  $\alpha$ -internal facts, formally  $s^\alpha = s \cap \mathcal{L}^\alpha$ . The set of *public actions* of agent  $\alpha$  is defined as  $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \text{atoms}(a) \cap \mathcal{L}^{\text{pub}} \neq \emptyset\}$  and *internal actions* as  $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$ . The symbol  $a^\alpha$  will denote a projection of action  $a \in \beta, \beta \neq \alpha$  for agent  $\alpha$ , i.e., action stripped of all other agents' propositions, formally  $\text{atoms}(a^\alpha) = \text{atoms}(a) \cap \mathcal{L}^\alpha$ .

Note that all actions of an agent  $\alpha$  uses only agent's facts, formally  $\forall a \in \alpha : \text{atoms}(a) \subseteq \mathcal{L}^\alpha$  by definition in (Brafman and Domshlak 2008). The goal set  $S_g$  of a multiagent planning problem will be treated as public (Nissim and Brafman 2012), therefore all goal-achieving actions are public. In the following sections, as an algorithm for multiagent planning, we will assume the MA-A\* from (Nissim and Brafman 2012), but with a novel distribution of the  $h^{\text{FF}}$  heuristic.

As a running example, we will use a simple logistics problem (see Figure 1) in a multiagent setting. There are two cities each with two locations A, B and C, D and one package  $p$ . A and D represent depots and B, C airports. Three agents represent two cargo trucks  $t_1, t_2$  (moving only within the cities) and one airplane  $a$  (moving only between airports B and C). The goal is to transport the package from depot A to the other depot D.

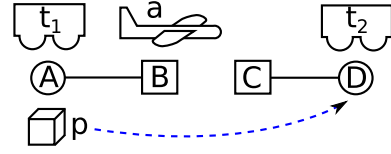


Figure 1: A running example is an instance of LOGISTICS problem with three agents and one package.

### Agent Relaxed Planning Graph

Relaxation is a way of simplifying a problem by removing some constraints. In planning, a relaxation is typically obtained by removing delete effect of actions. Solution of such relaxed planning problem is a relaxed plan, which can be used to estimate the cost of a plan in the original problem, e.g., the Fast-Forward heuristic estimation is based on the length of the relaxed plan. A classical technique for finding the relaxed plan is to build a Relaxed Planning Graph (RPG). RPG is a graph representing the reachability of facts and applicability of actions in the relaxed problem.

Building distributed planning graphs (not relaxed) was studied by (Pellier 2010), focusing on distribution of the Graphplan algorithm. Relaxed MA Planning Graphs were recently studied by (Torreño, Onaindia, and Sapena 2012), but in the area of planning with incomplete information and fluent cost estimation.

To obtain a more informed global heuristic estimate in a MA planning problem using the estimation based on a RPG, the RPG has to be decentralized. In this work, we propose a distributed global RPG in form of a set of distinct Agent RPGs. Such Agent RPG (ARPG) contains only facts of its owner agent. The initial state is projection for that agent and since the goals are treated as public, all agents have complete goals in their ARPGs. The usage of actions is straightforward in case of owner agent's internal and public actions which are used equally as in a classical RPG. Additionally, the Agent RPGs are extended by projections of other agents' public actions which were reachable by their particular owners. This extension enables the agents to take other agents' capabilities into account, but only at the time points, where their owners are able to reach them. Similarly to relaxed problems in STRIPS, we define a relaxed multiagent planning problem in MA-STRIPS as a problem stripped of delete effects in all actions of all agents:

**Definition 1.** An *agent relaxed planning graph (ARPG)* is a directed, labeled and layered graph  $\mathcal{R}^{\alpha} = (P' \cup A', E')$  of one particular agent  $\alpha$  for a relaxed multiagent planning task. Let  $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$  be a MA planning task, then a relaxed MA planning task  $\Pi' = (\mathcal{L}, \mathcal{A}', s_0, S_g)$  contains an altered set of agents  $\mathcal{A}'$ , s.t.,  $\forall \alpha \in \mathcal{A}$  and  $\alpha = \{a_1, \dots, a_{|\mathcal{A}|}\}$  there exist a relaxed agent  $\alpha' = \{a'_1, \dots, a'_{|\mathcal{A}|}\}$  and all its actions are relaxed versions of the regular actions  $a'_i = \langle \text{pre}(a_i), \text{add}(a_i), \emptyset \rangle$ . As in RPG, the nodes of the graph represent proposi-

tions  $P'$  and actions  $A'$ . The arcs  $E'$  represent linkup of propositions and actions.

In the rest of the paper, the discussion will be only about relaxed structures, therefore we will omit the prime signs, which are by convention used to denote relaxed structures.

ARPGs stem from the classical RPGs, therefore an  $i$ -th proposition layer and action layer will be denoted as  $P_i$  and  $A_i$  respectively. The layers alternate, so that  $(P_0, A_0, P_1, A_1, \dots, A_{n-1}, P_n)$  and all layers  $P_i \subseteq P$  and all layers  $A_i \subseteq A$ . The first proposition layer  $P_0$  contains nodes labeled by propositions of the agent's projection of the initial state, formally

$$P_0 = \{p | p \in s_0^\alpha\}.$$

Each action layer contains action nodes for all applicable relaxed actions of the agent  $\alpha$  in a state represented by the previous fact layer and external projections of other agents' public actions reachable in the same layer

$$A_i = \{a | a \in \alpha, \text{pre}(a) \subseteq P_i\} \cup \bigcup_{\beta \in \mathcal{A}, \beta \neq \alpha} \{b^\alpha | b \in P_i^\beta\}.$$

In all successive fact layers, the nodes copy the previous fact layer according to the frame axiom and transforms the facts by actions in the previous action layer, since for all relaxed actions  $\text{del}(a) = \emptyset$ , we can write

$$P_i = P_{i-1} \cup \{p | p \in \text{add}(a), a \in A_{i-1}\}.$$

At least one of the following terminating conditions has to hold for the last fact layer  $P_n$ :

- the last fact layer fulfills the goal  $S_G \subseteq P_n$ ,
- or  $P_n = P_{n-1}$ , meaning there are no additional actions which can extend further fact layers (a fixed-point).

The arcs in ARPG represent applicability and application of actions in relaxed states. We can split the arcs between two fact layers  $P_i$  and  $P_{i+1}$  into three groups. The first one contains arcs among facts of layer  $P_i$  and preconditions of actions in a layer  $A_i$ . The second one contains relation between effects of actions and next induced fact layer  $P_{i+1}$ . Additionally, there are arcs for all facts from a previous layer effectively representing the frame axioms of the closed world assumption. Formally,

$$\begin{aligned} E_i^{\text{pre}} &= \{(p_i, a_i) | p_i \in \text{pre}(a_i), a_i \in A_i\}, \\ E_i^{\text{add}} &= \{(a_i, p_{i+1}) | a_i \in A_i, p_{i+1} \in \text{add}(a_i)\}, \\ E_i^{\text{frm}} &= \{(p_i, p_{i+1}) | p_i \in P_i, p_{i+1} \in P_{i+1}, p_i = p_{i+1}\} \end{aligned}$$

and  $E_i = E_i^{\text{pre}} \cup E_i^{\text{add}} \cup E_i^{\text{frm}}$ . Now we will provide an algorithm for distributed building of ARPGs.

**Algorithm** The algorithm starts with each agent building an ARPG using only its own internal and public actions. An iterative process is then initiated, in which the agents exchange information about their *public* actions and extends their ARPGs with projected

**Algorithm 1** Distributed build of Agent Relaxed Planning Graphs

**Input:** An agent's factor of the relaxed MA planning problem  $\Pi^\alpha = \langle \mathcal{L}^\alpha, \alpha, s_0^\alpha, S_G \rangle$ .

**Output:** Agent Relaxed Planning Graph  $\mathcal{R}$  for  $\alpha$ .

---

```

1: init();
2:  $\mathcal{R} \leftarrow P_0 = \{p | p \in s_0^\alpha\}$ 
3:  $\mathcal{R} \leftarrow \text{build-RPG}(\mathcal{R}, \alpha, S_G)$ 
4:  $\mathcal{S} \leftarrow \text{map}[\alpha^{\text{pub}}, \text{integer}]$ 
5:  $\text{ack-count} \leftarrow \text{idle-count} \leftarrow 0$ 
6: check()

```

---

```

7: check();
8: for all  $a \in A_{n-1}$  s.t.  $a$  is public do
9:   if  $a \notin \mathcal{S}$  or  $\mathcal{S}[a] > \text{earliest layer of appearance of } a \text{ in } \mathcal{R}$  then
10:     $\mathcal{S}[a] \leftarrow \text{earliest appearance of } a \text{ in } \mathcal{R}$ 
11:     $\text{ack-count} \leftarrow \text{ack-count} + |\mathcal{A}|$ 
12:     $\forall \beta \in \mathcal{A} \setminus \alpha : \text{send}(\text{ext-a}[a^\beta, \mathcal{S}[a]], \text{to } \beta)$ 
13:   end if
14: end for

```

---

```

15: receive( $\text{ext-a}[a^\alpha, i \in \mathbb{N}]$ , from  $\beta \in \mathcal{A} \setminus \alpha$ ):
16:  $\forall \alpha \in \mathcal{A} : \text{send}(\text{not-idle}, \text{to } \alpha)$ 
17: send(ack, to  $\beta$ )
18:  $\mathcal{R} \leftarrow \text{extend-RPG}(\mathcal{R}, [a^\alpha, i])$ 
19:  $\mathcal{R} \leftarrow \text{build-RPG}(\mathcal{R}, \alpha, S_G)$ 
20: check()
21: goal-reached()

```

---

```

22: receive(ack):
23:  $\text{ack-count} \leftarrow \text{ack-count} - 1$ 
24: goal-reached()

```

---

```

25: receive(idle):
26:  $\text{idle-count} \leftarrow \text{idle-count} + 1$ 
27: if  $\text{idle-count} = |\mathcal{A}|$  then
28:   return  $\mathcal{R}$ 
29: end if

```

---

```

30: receive(not-idle):
31:  $\text{idle-count} \leftarrow \text{idle-count} - 1$ 

```

---

```

32: goal-reached();
33: if  $S_G \in \mathcal{R}$  and  $\text{ack-count} = 0$  and message queue is empty then
34:    $\forall \alpha \in \mathcal{A} : \text{send}(\text{idle}, \text{to } \alpha)$ 
35: end if

```

---

public actions of other agents. The algorithm terminates when the goal (or a fixed-point) is globally reached and there are no more messages to process.

The pseudo-code of the algorithm is given in Algorithm 1 in an event-driven fashion. The events can be caused either by receiving a message or internally by the algorithm itself. We assume that messages sent from one agent arrive in the same order as they were

1)													
a:	at-a-B	fly-a-B-C	at-a-B at-a-C	fly-a-C-B	at-a-B at-a-C								
t1:	at-p-A at-t1-A	load-t1-A drive-t1-A-B	at-p-A at-t1-A at-t1-B in-p-t1	load-t1-A unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1	load-t1-A load-t1-B unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1						
t2:	at-t2-D	drive-t2-D-C	at-t2-D at-t2-C	drive-t2-D-C drive-t2-C-D	at-t2-D at-t2-C								
2)													
a:	at-a-B	fly-a-B-C	at-a-B at-a-C	fly-a-C-B <b>unload-t1-B(t1)</b>	at-a-B at-a-C at-p-B	fly-a-C-B <b>unload-t1-B(t1)</b> load-a-B	at-a-B at-a-C at-p-B in-p-a	fly-a-C-B <b>unload-t1-B(t1)</b> load-a-B <b>unload-a-C</b>	at-a-B at-a-C at-p-B in-p-a				
t1:	at-p-A at-t1-A	load-t1-A drive-t1-A-B	at-p-A at-t1-A at-t1-B in-p-t1	load-t1-A unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1	load-t1-A load-t1-B unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1						
t2:	at-t2-D	drive-t2-D-C	at-t2-D at-t2-C	drive-t2-D-C drive-t2-C-D <b>unload-t1-B(t1)</b>	at-t2-D at-t2-C								
3)													
a:	at-a-B	fly-a-B-C	at-a-B at-a-C	fly-a-C-B unload-t1-B(t1)	at-a-B at-a-C at-p-B	fly-a-C-B unload-t1-B(t1) load-a-B	at-a-B at-a-C at-p-B in-p-a	fly-a-C-B unload-t1-B(t1) load-a-B unload-a-B <b>unload-a-C</b>	at-a-B at-a-C at-p-B in-p-a				
t1:	at-p-A at-t1-A	load-t1-A drive-t1-A-B	at-p-A at-t1-A at-t1-B in-p-t1	load-t1-A unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1	load-t1-A load-t1-B unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A	at-p-A at-p-B at-t1-A at-t1-B in-p-t1	load-t1-A load-t1-B unload-t1-A <b>unload-t1-B</b> drive-t1-A-B drive-t1-B-A <b>unload-a-C(a)</b>	at-p-A at-p-B at-t1-A at-t1-B in-p-t1				
t2:	at-t2-D	drive-t2-D-C	at-t2-D at-t2-C	drive-t2-D-C drive-t2-C-D unload-t1-B(t1)	at-t2-D at-t2-C	drive-t2-D-C drive-t2-C-D unload-t1-B(t1)	at-t2-D at-t2-C	drive-t2-D-C drive-t2-C-D unload-t1-B(t1) <b>unload-a-C(a)</b>	at-t2-D at-t2-C <b>at-p-C</b>	drive-t2-D-C drive-t2-C-D unload-t1-B(t1) <b>unload-a-C(a)</b> load-t2-C	at-t2-D at-t2-C <b>at-p-C</b> in-p-t2 <b>unload-t1-B(t1)</b> <b>unload-a-C(a)</b> load-t2-C <b>unload-t2-D</b>	drive-t2-D-C drive-t2-C-D unload-t1-B(t1) <b>unload-a-C(a)</b> load-t2-C <b>at-p-D</b> in-p-t2	at-t2-D at-t2-C

Figure 2: Distributed building of Agent Relaxed Planning Graphs decomposed into iterations.

sent, but we assume no ordering between messages sent from different agents. We will now explain each event handling routine.

In the **init** phase a Relaxed Planning Graph  $\mathcal{R}$  is built using only agent's own actions by **build-RPG** method from the initial state projection  $s_i^0$ . A map  $\mathcal{S}$  used to store the *earliest layer of appearance*<sup>1</sup> of the agent's public actions is initialized along with other supporting data structures used for synchronized termination of the algorithm. After the initialization phase, reaching of the goal (or a fixed-point) is checked, and if positive, all agents are informed that the agent is idle now. Next, the executed **check** procedure is responsible for checking whether  $\mathcal{R}$  contains any public actions. If so, each action is sent to all other agents  $\beta \in \mathcal{A} \setminus \alpha$  as a projection  $a^\beta$  with its earliest layer of appearance, unless it was already broadcasted with equal or lower number of layer (this can happen in future **check** calls).

<sup>1</sup>Earliest layer of appearance of an action  $a$  in (A)RPG is the first action layer, where  $a$  is applicable.

In the next part of the algorithm, there are four message handling **receive** procedures. The first one *ext-a* is executed when a projection of other agent's public action is received. After sending control messages, the action is integrated into  $\mathcal{R}$  on the  $i$ -th layer by **extend-RPG** method and the change is propagated by **build-RPG**, so that all actions newly applicable in the following layers are applied accordingly. Then the built ARPG is checked, whether new public actions (and public actions newly applicable on earlier layers) are reachable and whether the goal or the fixed-point was reached. The last three **receive** procedures maintain the control information needed for distributed termination detection (Mattern 1987). The *acks* counter keeps track of number of sent external actions and postpones termination until all sent actions are processed. If an *idle* message is received, there are no pending *acks* and the number of idle agents is equal to  $|\mathcal{A}|$ , the algorithm terminates and the resulting ARPG  $\mathcal{R}$  is returned. Since *not-idle* and *ack* messages are sent in this particular order (lines 17 and 18) and the messages from one agent

are presumed to keep ordering, the algorithm terminates synchronously when all external actions are processed and no messages are pending.

In Figure 2, the Algorithm 1 is applied on the running example depicted in Figure 1. Although the algorithm is running asynchronously, we can decompose it for clarity into several iterations. In the first iteration, the ARPGs are built using only the actions of the respective agents  $\mathbf{a}$ ,  $\mathbf{t}_1$  and  $\mathbf{t}_2$  (airplane and two trucks). Notice the bold green action **unload-t1-B**, which is a public action of the truck  $\mathbf{t}_1$ , can be applied thanks to the initial position of the package. In the next iteration, projection of the public action is broadcasted and received by other agents. Upon receiving, their ARPGs are updated, which for the airplane means that the ARPG is expanded with further layers. Another public action **unload-a-C** is applied and therefore broadcasted. In the third iteration, the projection of the airplane’s unload action is added to the ARPGs of the trucks. For truck  $\mathbf{t}_1$  it has no effect, but it allows truck  $\mathbf{t}_2$  to expand the ARPG and reach goal **at-p-D**. Notice, that when the projected **unload-a-C(a)** was received by truck  $\mathbf{t}_2$ , its ARPG was first extended to have enough layers for the action to be added to the correct layer.

Although not shown in Figure 2, the algorithm would continue with one more iteration after broadcasting the public action reached by truck  $\mathbf{t}_2$ , resulting in all agents having ARPGs with the same number of layers and all having reached the goal. Additionally, the algorithm does not have to terminate when the goal is reached, but can continue until the fixed-point, which can be desirable in some situations and which is also the case when the goal is not reachable.

**Proof sketch** In this section, we will sketch a proof showing, that the Agent Relaxed Planning Graphs built by Algorithm 1 are *compatible* with a global RPG, meaning that they contain the same actions (with respect to projections) and that the actions are in the same layers. We will use this proven theorem further in a proof of equality of the centralized FF and multiagent FF (MAFF) heuristics.

Firstly, we will formally define the concept of compatibility, then we will show that single iteration of the algorithm does not violate the compatibility, and finally we will show that the algorithm terminates, i.e., the resulting ARPGs are compatible with a centrally built RPG and that no actions are missing or are superfluous.

Let  $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_G \rangle$  be a relaxed MA planning task,  $\mathcal{R}^\alpha$  be an Agent Relaxed Planning Graph for agent  $\alpha$  built from  $\Pi$  using Algorithm 1, having alternating layers  $(P_0^\alpha, A_0^\alpha, P_1^\alpha, A_1^\alpha, \dots, A_{n-1}^\alpha, P_n^\alpha)$  and let  $A^\alpha = \bigcup_{i \in \langle 0, n-1 \rangle} A_i^\alpha$  and  $A^A = \bigcup_{\alpha \in \mathcal{A}} A^\alpha$ . Let  $\hat{\Pi} = \langle \mathcal{L}, \text{Act}, s_0, S_G \rangle$  be a classical relaxed planning task,  $\hat{\mathcal{R}}$  be a classical Relaxed Planning Graph built from  $\hat{\Pi}$  having alternating layers  $(\hat{P}_0, \hat{A}_0, \hat{P}_1, \hat{A}_1, \dots, \hat{A}_{n-1}, \hat{P}_n)$  and let  $\hat{A} = \bigcup_{i \in \langle 0, n-1 \rangle} \hat{A}_i$ . Note that from the mono-

tonicity of (A)RPGs follows  $\forall i < j : P_i \subseteq P_j$  and  $\forall i < j : A_i \subseteq A_j$ .

**Definition 2.** Let  $a \triangleright A_i$  denote that an action  $a$  is *first applicable* in layer  $A_i$  (formally  $\text{pre}(a) \subseteq P_i \wedge \text{pre}(a) \not\subseteq P_{i-1}$ ) regardless of whether the underlying structure is RPG or ARPG.

**Definition 3.** We define that a set of ARPGs  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  is *compatible* with a RPG  $\hat{\mathcal{R}}$  iff for each action  $a \in A^A$  for which  $a \triangleright \hat{A}_i$  holds the following:

- 1) If  $a$  is an *internal action* of agent  $\alpha$ , then  $a \triangleright A_i^\alpha$ .
- 2) If  $a$  is a *public action* of agent  $\alpha$ , then  $a \triangleright A_i^\alpha$  and  $\forall \beta \in \mathcal{A} \setminus \alpha : a^\beta \triangleright A_i^\beta$ , where  $a^\beta$  is the projection of action  $a$  for agent  $\beta$ .

**Lemma 4.** A set of ARPGs  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  compatible with a RPG  $\hat{\mathcal{R}}$  stays compatible with  $\hat{\mathcal{R}}$  after application of *build-RPG* by agent  $\alpha$  and *successive extend-RPG* by all other agents.

*Proof.* Let us have a RPG  $\hat{\mathcal{R}}$  built from a relaxed planning task  $\hat{\Pi}$  and a set of ARPGs  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  being built from relaxed MA planning task  $\Pi$  using Algorithm 1. The symbol  $A^A$  denotes all actions applied in the algorithm so far and let  $\alpha^{\text{proj}}$  be the set of projected actions received by agent  $\alpha$  so far. Now, agent  $\alpha$  applies *build-RPG*, so that  $\mathcal{R}^\alpha$  is updated by  $A_i^\alpha = A_i^\alpha \cup \{a | a \in \alpha \cup \alpha^{\text{proj}} : \text{pre}(a) \subseteq P_i^\alpha\}$  (and accordingly  $P_{i+1}^\alpha$ ), for each layer  $A_i^\alpha$ . Let us assume, there exists an extra action  $a \in \alpha$  which was newly applied ( $a \notin A^A$ ) and for which  $a \triangleright \hat{A}_i \wedge a \not\triangleright A_i^\alpha$ . We can also assume WLOG, that  $a$  is first such action (in terms of layer of appearance).

From definition of  $a \triangleright \hat{A}_i$ , where  $\text{pre}(a) \subseteq \hat{P}_i \wedge \text{pre}(a) \not\subseteq \hat{P}_{i-1}$  follows that  $\text{pre}(a) \subseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-1}, p \in \text{add}(b)\}$  and  $\text{pre}(a) \not\subseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-2}, p \in \text{add}(b)\}$ . Because  $a$  is first action for which  $a \triangleright \hat{A}_i \wedge a \not\triangleright A_i^\alpha$ , for all actions  $b \in \alpha$ , for which holds  $b \triangleright \hat{A}_k$  where  $k < i$ , holds also  $b \triangleright A_k^\alpha$ . Therefore  $A_k^\alpha = \hat{A}_k \cap \alpha$  and  $P_k^\alpha = \hat{P}_k \cap \text{atoms}(\alpha)$  for all  $k < i$  and therefore  $\text{pre}(a) \subseteq P_i^\alpha \wedge \text{pre}(a) \not\subseteq P_{i-1}^\alpha$ , which means  $a \triangleright A_i^\alpha$  and that is a contradiction. Now, we can assign  $A^A \leftarrow A^A \cup \{a\}$  and repeat the former step.

After broadcasting projections of the newly applied public actions and calling *extend-RPG* by all other agents, we can show that the second part of Definition 3 also holds. Let  $a^\alpha$  be the projection of an action  $a \in \beta$  which is broadcasted first. If there exists some  $i$  for which  $a \triangleright \hat{A}_i$  then  $\text{pre}(a) \subseteq \hat{P}_i$  and for the projection  $a^\alpha$  holds  $\text{pre}(a) \subseteq \hat{P}_i \cap \mathcal{L}^{\text{pub}}$ . Because for all actions  $b \in \hat{A}_{i-1}$  the lemma holds, if  $b$  is public,  $b^\alpha \triangleright A_{i-1}^\alpha$ . Because  $a^\alpha$  is a projection,  $\text{pre}(a^\alpha) \subseteq P_0^\alpha \cup \bigcup_{b^\alpha \in A_{i-1}^\alpha} \text{add}(b)$  and therefore  $a^\alpha$  is applicable in layer  $i$  (and subsequent layers), which means that the *extend-RPG* ensures that  $a^\alpha \triangleright A_i^\alpha$ .  $\square$

**Theorem 5.** *When Algorithm 1 terminates, resulting set of ARPGs  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  built from  $\Pi$  is compatible with the RPG  $\hat{R}$  built from  $\hat{\Pi}$  and there are no additional actions, i.e.,  $\hat{A} = A^A$ . Each public action in  $\hat{A}$  has its projected counterparts in  $A^A$  and vice versa, i.e., for each public action  $a \in \hat{A}$  such that  $a \in \alpha$  for some agent  $\alpha$  exists projected action  $a^\beta$  for each agent  $\beta \in \mathcal{A} \setminus \alpha$  and  $a^\beta \in A^A$ . There is no projected action  $a^\beta \in A^A$  for which there is no original action  $a \in \hat{A}$ .*

*Proof.* We will now sketch an induction which shows the compatibility in Theorem 5, based on the Lemma 4. For the initial step of the induction we take all ARPGs containing only the first fact layer  $P_0^\alpha$  which is trivially compatible because  $A^A = \emptyset$ . The induction step is covered by Lemma 4, because each step of the Algorithm 1 can be decomposed as an application of build-RPG and, if there are any applied public actions, broadcasting their projections and application of extend-RPG by all other agents. Even though the algorithm is running asynchronously, Lemma 4 holds because of the monotonicity of (A)RPGs.

Termination of the algorithm follows from the termination of classical RPG, either the algorithm reaches goal or a fixed-point, where no more actions are added. Similarly to classical RPG, building of ARPGs is monotonic, which means that the facts and actions can only be added and because the set of actions is finite, there must be a point where no more actions can be added and the algorithm terminates. The detection of such situation is more complicated in the distributed setting and is described thoroughly in the algorithm section.

The last statement we are about to show is that there are no additional actions, i.e.,  $\hat{A} = A^A$  (irrespective of the projections of public actions) and that each public action in  $\hat{A}$  has its projection in  $A^A$  and vice versa. Let us assume, that  $\exists a \in \hat{A}$  such that  $a \notin A^A$ , let us also assume, WLOG, that  $a$  is such action appearing in the earliest layer in  $\hat{R}$ , say  $\hat{A}_i$ , and that  $a \in \alpha$  for some agent  $\alpha$ . We know, that exists minimal  $A_{\text{pre}} \subseteq \hat{A}_{i-1}$  such that  $\text{pre}(a) \subseteq \{p | b \in A_{\text{pre}}, p \in \text{add}(b)\}$ , because  $\text{pre}(a) \subseteq \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$ , for all actions  $b \in A_{\text{pre}}$  either  $b \in \alpha$  or  $b$  is public. Because we assumed, that  $A_{\text{pre}} \subseteq \hat{A}$ , for each  $b \in A_{\text{pre}}$ , if  $b \in \alpha$  then  $b \in A_{i-1}^\alpha$  and if  $b \notin \alpha$  then  $b$  is public, therefore  $b^\alpha \in A_{i-1}^\alpha$ . From the said  $\text{pre}(a) \subseteq \{p | b \in A_{i-1}^\alpha, p \in \text{add}(b)\}$ , which means that  $a$  is applicable in  $A_i^\alpha$  and therefore  $a$  must be applied by the algorithm. If we continue with next such action we end up with  $\hat{A} \subseteq A^A$ .

Now, we will show that  $\hat{A} \supseteq A^A$ . Let us assume, that  $\exists a \in A^A$  such that  $a \notin \hat{A}$  and that  $a$  is first such action. Similarly to the previous situation, if  $a$  is applicable in some layer  $A_i^\alpha$  then there is some set of actions  $A_{\text{pre}} \subseteq A_{i-1}^\alpha$ , which contains all actions providing preconditions of  $a$ . Since all actions  $b \in A_{\text{pre}}$  are also in  $\hat{A}$ ,  $a$  is applicable in  $\hat{A}_i$  and therefore must be applied.

From  $\hat{A} \subseteq A^A$  and  $\hat{A} \supseteq A^A$  follows that  $\hat{A} = A^A$ . We have already shown that public actions have their projected counterparts. The only remaining part to show is that there are no projected actions in  $A^A$  without their respective original actions in  $\hat{A}$ . This clearly follows from the algorithm itself, because all projected actions are created only when a public action is added to some  $A_i^\alpha$  by agent  $\alpha$  and as shown before, such action would also be added to  $\hat{A}_i$ .  $\square$

## Multiagent FF Heuristic

With the help of ARPGs, the Fast-Forward heuristic estimate can be straightforwardly adapted to a multiagent setting. We will denote such heuristic as  $h^{\text{MAFF}}$ . The multiagent (MA) relaxed plan backing the  $h^{\text{MAFF}}$  estimate can be in general spread over all ARPGs of the agents in the team as illustrated in Figure 3. The most left achieving actions has to be considered from all agents. In the case of projected public actions, the owner agent has to define part of the relaxed plan, possibly using his internal actions, to achieve the internal facts of the provided public action. Additionally, the relaxed plan has to share public actions which are required by more agents at the same layers. The private parts of the relaxed plan provided by the other agents can be described by place-holding actions and therefore no private information of the other agents has to be revealed. The final heuristic estimate is the count of actions of the MA relaxed plan.

**Definition.** Let a MA relaxed plan  $\pi$  be a solution of a MA relaxed problem  $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$ , where  $s$  is the state, we are estimating the cost for, then  $|\pi| = h(s)$  is the *multiagent relaxation heuristic estimate*.

Similarly to the relaxation heuristic estimate  $h^{\text{FF}}$ , we restrain  $\pi$  for  $h^{\text{MAFF}}$  according to  $h^{\text{FF}}$ . A particular  $\pi$  is defined using ARPGs  $\mathcal{R}^\alpha = (P \cup A, E)$  of all agents  $\alpha \in \mathcal{A}$  built for the state  $s$ . From the right (meaning as in Figure 3), the relaxed plan  $\pi$  contains minimal set of actions  $A_m^* \subseteq A_m$ , achieving the goal facts. The action layer  $A_m$  contain actions of all agents in layer  $m$  (ignoring projections of actions, since the respective original actions are also included in the same layer). If there is a frame arc  $(p_{m-1}, p_m) \in E_m^{\text{frm}}$  of such facts, i.e.,  $p_m \in S_G$ , the fact  $p_m$  does not need an explicit achieving action from this particular layer as it will be achieved by an action from an earlier (more left) layer. This principle effectively selects the most-left achievers of a fact as proposed by FF heuristic. The action set  $A_m^*$  induces next set of facts across all agents

$$S_m = \{p | p \in \text{pre}(a) : a \in A_m^*\},$$

which has to be achieved by actions from previous action layer  $A_{m-1}$  and so on until the action layer  $A_0$  is reached, where all the actions have their preconditions satisfied by the initial state in  $P_0$ .

Notice that since this definition works across all ARPGs of all agents, the resulting  $\pi$  may contain actions of different agents.



a:	at-a-B	fly-a-B-C	at-a-C	unload-t1-B(t1)	at-p-B	load-a-B	in-p-a	unload-a-C	at-a-C						
t1:	at-p-A	load-t1-A	at-t1-B	unload-t1-B	at-p-B										
	at-t1-A	drive-t1-A-B	in-p-t1												
t2:	at-t2-D	drive-t2-D-C	at-t2-C					unload-a-C(a)	at-p-C	load-t2-C	in-p-t2	unload-t2-D	at-p-D		

Figure 3: Multiagent Relaxed Plan

We can compute  $h^{\text{MAFF}}(s)$  by first building the set of ARPGs  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  for relaxed MA planning problem  $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$  using Algorithm 1, then simultaneously extracting relaxed plans  $\pi_\alpha$  for each agent using Algorithm 2 and finally summing the lengths of the resulting relaxed plans, excluding projections of other agent’s public actions.

**Theorem 6.** *Let  $\pi_\alpha \cap \alpha$  be the computed relaxed plan of agent  $\alpha$  restricted only to the agent’s actions (excluding all projections of other agent’s public actions), then  $h^{\text{MAFF}}(s) = \sum_{\alpha \in \mathcal{A}} |\pi_\alpha \cap \alpha| = h^{\text{FF}}(s)$ .*

*Proof.* The fact that  $h^{\text{MAFF}}(s) = h^{\text{FF}}(s)$  follows from the previously shown compatibility of the RPG  $\hat{\mathcal{R}}$  built for relaxed planning problem  $\hat{\Pi}$  and the set of ARPGs built from  $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$  for relaxed MA planning problem  $\Pi$ . We first extract relaxed plans  $\hat{\pi}$  for  $\hat{\Pi}$  and  $\{\pi_\alpha | \alpha \in \mathcal{A}\}$  for  $\Pi$  by effectively choosing first achievers of goal facts and of preconditions of previously chosen achievers. It is clear that for some fact  $p$  we choose an action  $a$  s.t.  $p \in \text{add}(a)$  only if  $a \triangleright A_i$  for some layer  $A_i$  and there is no action  $b$  s.t.  $p \in \text{add}(b)$  and  $b \triangleright A_j$  for some  $j < i$ . Because of the compatibility of the RPG and ARPGs, we choose exactly the same actions (and their projections, which are then omitted) for  $\hat{\pi}'$  and for  $\{\pi'_\alpha | \alpha \in \mathcal{A}'\}$ , which means that  $|\hat{\pi}'| = \sum_{\alpha \in \mathcal{A}'} |\pi'_\alpha \cap \alpha|$  and therefore  $h^{\text{MAFF}}(s) = h^{\text{FF}}(s)$ .  $\square$

## Experiments

The experiments were conducted on an implementation of satisficing version of MA-A\* (Nissim and Brafman 2012) with various relaxation heuristic estimates<sup>2</sup>. The algorithm begins with a centralized factorization and a reachability analysis of a centralized planning problem. After the factorization, the agents are started receiving its factors of the problem as an input. The agents run in parallel, each one in its own thread and the messages are delivered by an additional asynchronous messaging thread. Each agent uses an event queue to serialize the computation and reactions to incoming messages. If an agent finds a sound plan (with parts from other agents) it prints it and stops the distributed process.

Since the algorithm is asynchronous, the runs are non-deterministic. Therefore we conducted each experimental run as ten measurements. Each measurement was limited to 8GB of memory for the Java Virtual Machine and to 10 minutes of runtime. Each measurement

<sup>2</sup>Technically the implementation is not A\* as the heuristics are not admissible, therefore the used algorithm is precisely MA-BestFirstSearch.

---

### Algorithm 2 Distributed extraction of $h^{\text{MAFF}}$

---

**Input:** ARPG  $\mathcal{R}$  for state  $s$ , having layers  $(P_0, A_0, P_1, A_1, \dots, A_{n-1}, P_n)$  and goal  $S_G \subseteq \mathcal{L}$ .

**Output:** Relaxed plan  $R$  from  $s$  to  $S_G$ .

---

```

1:  $P \leftarrow S_G$ 
2:  $R \leftarrow \emptyset$ 
3: for  $i = n - 1; i > 0; i \leftarrow i - 1$  do
4:    $P' \leftarrow \emptyset$ 
5:   for  $p \in P$  do
6:     if  $p \notin P_i$  then
7:        $a \leftarrow a \in A_i$ , such that  $p \in \text{add}(a)$ 
8:        $R \leftarrow R \cup \{a\}$ 
9:        $P' \leftarrow P' \cup \text{pre}(a)$ ,  $P \leftarrow P \setminus \{p\}$ 
10:      if  $a$  is projected action of agent  $\alpha$  then
11:        request  $R^\alpha$  for  $s$ , goal  $\text{pre}(a^{\text{orig}})$  from  $\alpha$ 
12:         $R \leftarrow R \cup R^\alpha$ 
13:      end if
14:    end if
15:  end for
16:   $P \leftarrow P \cup P'$ 
17: end for
18: return  $R$ 

```

---

was run on 8-core processor at 3.6GHz separately. The results from the measurements were averaged.

We used five planning domains, four originating in the single-agent IPC planning benchmarks. Similarly to the evaluation of the algorithms in (Nissim, Brafman, and Domshlak 2010), we chose domains which are straightforwardly modifiable to the multiagent setting: LOGISTICS (similar to the running example, but with more agents), LINEAR LOGISTICS (one package has to be transported stepwise by all agents), ROVERS, and SATELLITES. As in (Komenda, Novák, and Pěchouček 2013), we have extended the set of IPC-based domains by a coordination domain COOPERATIVE PATHFINDING, in which robots on a grid are tasked to switch their positions not colliding with each other. We tested following distribution strategies of FF heuristic estimation:

- $h_\alpha^{\text{FF}}$  using only locally built RPGs (including projections of public actions) and local estimation of Fast-Forward heuristic, as proposed in (Nissim and Brafman 2012).
- $h^{\text{MAFF}}$  using distributed ARPGs based on Algorithm 1 and distributed extraction of FF heuristic as described in Algorithm 2.
- Lazy  $h^{\text{MAFF}}$  using only locally built RPGs (including projections of public actions) and distributed extrac-

			$h_{\alpha}^{\text{FF}}$				$h^{\text{MAFF}}$						Lazy $h^{\text{MAFF}}$					
	$ \mathcal{A} $	$l^*$	$t[s]$	$v$	$c_s$	$l$	$t[s]$	$v$	$c_s$	$c_r$	$c_h$	$l$	$t[s]$	$v$	$c_s$	$c_r$	$c_h$	$l$
CP	3	6	0.4	99	97	6	6.3	168	166	6.6k	39	6	0.7	117	115	72	40	6
	5	16	88	25k	25k	16	–	–	–	–	–	–	7.6	1.8k	1.8k	120	145	18
	7	24	–	–	–	–	–	–	–	–	–	–	323	52k	52k	168	254	30,9
LOG	4	14	0.6	1.5k	847	14	2.5	505	272	10k	50	14	0.4	365	223	32	45	14
	6	20	6.2	17k	7.4k	20,6	–	–	–	–	–	–	1.3	732	341	52	185	20
LLG	6	18	0.2	134	61	18	4.9	118	54	2.0k	35	18	0.5	130	61	22	21	18
	8	24	0.6	241	113	24	11	216	102	4.6k	64	24	1.2	231	108	30	34	24
	10	30	0.9	381	181	30	27	337	161	8.7k	99	30	2.8	357	170	38	46	30
ROV	2	22	–	–	–	–	88	378	4	25k	4	22	19	482	4	72	4	22
	3	33	–	–	–	–	–	–	–	–	–	–	261	2.0k	14	108	11	33
SAT	4	14	32	32k	369	14	16	941	27	9.8k	22	14,3	0.8	536	29	4	23	14,3
	6	21	–	–	–	–	–	–	–	–	–	–	6.2	1.7k	67	6	54	21,3
	8	–	–	–	–	–	–	–	–	–	–	–	45	4.5k	147	8	104	28,1

Table 1: Experimental results for the heuristics.  $|\mathcal{A}|$  is number of agents,  $l$  is sequential length of the plan ( $l^*$  is optimal),  $t$  is duration of the search in seconds,  $v$  is a number of visited states,  $c_s$  is a number of search messages (each of size of a state),  $c_r$  is a number of messages building ARPGs (each of size of a projected public action) and  $c_h$  is a number of messages for the heuristic estimate (each of size of a partial relaxed plan). As  $h_{\alpha}^{\text{FF}}$  do not build distributed ARPGs,  $c_r$  and  $c_h$  are always zero. The domains are COOP. PATHFINDING (CP), LOGISTICS (LOG), LINEAR LOGISTICS (LLG), ROVERS (ROV) and SATELLITES (SAT). Runs denoted as – did not finish in the limits.

tion of FF heuristic as in Algorithm 2 with additional information on reachability of projected actions.

The implementation we used is a preliminary prototype which is not competitive with the current single-agent state-of-the-art planners, but nevertheless it gives insights in comparison of the heuristics.

The  $h^{\text{MAFF}}$  is based on the theory presented in the previous section. For each state each of the agents computes the heuristic estimate, i.e., a complete set of ARPGs is built. In order to manage the distribution and asynchronism, the ARPG building algorithm slightly differs from Algorithm 1 so that ARPGs for several different states can be built simultaneously. The heuristic estimate is then extracted using Algorithm 2.

In Lazy  $h^{\text{MAFF}}$ , the ARPGs are built lazily, i.e., an agent builds an ARPG for its current search state using its actions and projections similarly as in  $h_{\alpha}^{\text{FF}}$ , then the Relaxed Plan (RP) is extracted and only when some projected action is added to the RP, request is sent to the owner of the original action. The owner then builds an ARPG from the given state to preconditions of the original actions, extracts a RP using the same procedure and sends back the computed RP. The returned RP is then merged with the original one. This effectively forms a distributed recursion algorithm. Such algorithm significantly lowers communication load and enables the agents to search in parallel, especially in loosely coupled problems. In addition to this, reachability analysis using Algorithm 1 can be done before starting the search to improve the estimate of applicability of the projected actions.

The results in Table 1 show, that  $h_{\alpha}^{\text{FF}}$  is fast and can effectively solve smaller problem instances, but it is not

much informed, as illustrated by the number of visited states. This becomes critical in larger problems. On the other hand,  $h^{\text{MAFF}}$  is better informed, but it has to build all ARPGs for each state which is estimated. This is extremely communication intensive as shown by the number of exchanged ARPG messages ( $c_r$ ). Also the possibilities of parallel computation are reduced by the fact that all agents have to build the ARPGs for each estimated state.

The best performance is given by the Lazy  $h^{\text{MAFF}}$ . This implementation keeps the heuristic estimate quality of  $h^{\text{MAFF}}$ , but since it does not build ARPGs during the search, but only local RPGs enriched by the projections of other agents' actions, the RPGs can be built lazily only for those states where any interaction between the agents is needed and only those agents involved build the RPGs. The ARPGs are computed only in an initial reachability analysis, which can be omitted, but which significantly improves the results.

## Final Remarks

Our formal treatment and design of algorithms for computing distributed Relaxed Planning Graph for multi-agent planning can be seen as a first step towards efficient MA planners based on satisficing state-space search techniques utilizing relaxation heuristics. Furthermore, we have experimentally shown that appropriate implementation of a decentralized estimation of a global relaxation heuristic can radically improve computational and communication efficiency of the planning process as a whole.

**Acknowledgments** This work was supported by the *U.S. Air Force EOARD* grant no. FA8655-12-1-2096.

## References

- Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of ICAPS'08*, 28–35. AAAI.
- Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of ICAPS'09*. AAAI.
- Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of ICAPS'07*, 176–183. AAAI.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2012.12.011.
- Mattern, F. 1987. Algorithms for distributed termination detection. *Distributed computing* 2(3):161–175.
- Nissim, R., and Brafman, R. I. 2012. Multi-agent A\* for parallel and distributed systems. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *Proceedings of AAMAS'12*, 1265–1266. IFAAMAS.
- Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS'10*, 1323–1330.
- Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *Proceedings of ICAART'10*, volume 2, 128–134. IFAAMAS.
- Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *Proceedings of ECAI'12*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 762–767. IOS Press.